C++ 프로그람설계

프로그람작성 및 객체지향설계

평양콤퓨터기술대학
외국문도서출판사
주체 91

C++프로그람설계

프로그람작성 및 객체지향설계

평양콤퓨터기술대학
외국문도서출판사

차 례

제6장. 프로그람작성자정의함수

제7장. 클라스구조와 객체지향설계

제8장. 추상자료형의 실현

제9장. 목록

제10장. EzWindows API의 구체적인 조사

머리말

소개

콤퓨터는 우리 생활에서 필수적인것으로 되였다. 콤퓨터는 망에서 재정관리, 정보전송, 전화망체계, 동력설비와 같은 복잡한 체계들을 조종한다. 오늘 수많은 사람들이 인터네트상에서 콤퓨터를 리용하여 관광, 상점에 대한 정보를 보고 있다. 그러므로 누구나 현대생활의 기본구성요소인 콤퓨터로 프로그람을 작성하는 방법을 알아야 할 필요가 제기되고 있다.

이 교재에서는 프로그람작성의 기초적인 문제와 C++로 체계프로그람을 개발하는 공정을 서술하였다. 객체지향에 의한 체계를 개발하기 위하여 C++언어를 소개한다. 이 책에서는 초보적인 프로그람작성과 련습을 통하여 어느 정도 프로그람을 작성할수 있게 해준다. 대학1학년학생들의 수준에서 특별한 프로그람작성경험이 없고 일반수학적지식만을 가지고 있다는 점을 고려하였다.

이 책의 특징은 다음과 같다.

- 응용범위가 대단히 큰것이다. C++에서 개선된 기능을 보여 준다. 초학자들에게는 약간 힘들다. 왜냐하면 이 언어가 150개이상의 표준클라스와 서고를 정의하고 있기때문이다. 이 책에서는 처음에 필요한 지식과 구체적인 내용, 지적자에 대하여 깊이 있고 구체적인 설명을 주었다. C++는 표준화서고, 이름공간, 례외처리형 Bool과 같은 넓은 범위의 기술을 제공한다. 이 책은 대학생들의 참고서로서 C++언어를 학습하는데 큰 도움을 줄것이다.
- 클라스에 대하여 소개하였다. 4장은 객체지향의 개념에 대하여 보여 주었다. 객체를 리용하여 체계를 설계하기전에 먼저 학생들이 객체에 대한 개념을 가지고 있지 않다고 보고 설명하였다. 이미 나온 프로그람실례들도 처음에는 사용자의 립장에서부터 시작하였다. 다음장들에서 표준서고, 도형처리서고에서 파생된 객체들가운데서 cout, cin, string에 대하여 소개하고 그 리용방법을 설명하였다. 이 책은 객체지향의 개념, 쏘프트웨어의 재리용, 정보은페의 개념을 리해하는데 큰 도움을 줄것이다. 1장에서는 객체에 대하여 간단히 서술하고 그다음 8개의 장에서 약 50개의 클라스들과 추상자료형(ADT)들에 대하여 설명하였다.
- 초학도들이 재미 있는 프로그람을 작성할수 있도록 개발된 API를 소개한다. 간단한 도형, 그림, 본문객체를 쉽게 현시할수 있는 EzWindows라는 객체지향도형서고에 대해서도 설명하였다. 또한 Windows와 UNIX콤파일러의 API실행에 대해서도 보여 준다. API를 리용하면 학생들은 아주 중요한 경험을 체득할수 있다. 학생들은 처음 서고에 대하여 사용자적견지에 있어야 한다. 앞에서 언급한바와 같이 잘 설계된 객체를 리용하여 초학자들이 객체지향프로그람의 설계방법을 쉽게 리해할수 있게 하였다. 프로그람설계를 잘할수 있는 기초는 사용자적견지에서 많은 경험을 쌓는것이다. 또한 지정된 서고에서 작성된 응용프로그람을 개발할수 있는 능력을 키우는 것이다. 도형의 입출력을 수행하는 EzWindows를 리용하여 사건구동형프로그람과 실지응용프로그람에서 기능이 높은 입출력방법을 보여 준다. EzWindows의 리용방법에 대해서는 책의 여러 부분에서 설명하였다. 높은 도형처리능력을 요구하는 학생들과 교원들은 부록1을 보면 EzWinodws의 특징을 구체적으로 알수 있다. 상세한 내용은 http://www.cplusplus-programdesign.com을 통해서 알아 보기를 바란다.
- 프로그람과 쏘프트웨어개발을 학습하여 쏘프트웨어개발공학설계의 개념을 리해할수 있게 하였다. 매장에서는 구체적인 실례를 들어 C++와 객체지향설계의 개념, 중점적인 문제들을 상세하게 서술하였다. 중점적인 문제는 객체지향분석, 설계, 알고리듬개발, 설계프로그람작성과 같은 것들이다. 10장과 15장에서는 EzWindows API를 리용한 쏘프트웨어프로그람의 개발원리를 설

명하였다. 이 장들에서는 개별적 혹은 묶음별일감과 프로그람의 재리용에 대하여 구체적으로 설명하였다.

- 프로그람작성과 경험들을 명백하게 구분하였다. 높은 기능의 C++와 객체지향프로그람작성법을 설명하기 위하여 프로그람작성자들과 설계가들이 알아야 할 문제들을 구체적으로 설명하였다. 실례로 프로그람작성자들속에서 제기되는 방법론적문제들을 설명하였다. 이러한 내용들을 《프로그람작성경험》과 《계산의 력사》 등의 술어를 붙여 매장에서 1~2페지정도 설명하였다.
- 통합된 표준본보기서고의 리용에 대하여 서술하였다. 표준본보기서고는 C++언어의 중요한 구성부분이다. 보통 STL이라고도 한다. 이 서고는 프로그람작성자들이 검색, 분류, 목록의 삽입과 같은데서 제기되는 기술적인 문제들인 알고리듬, 문자렬, 목록을 표시하기 위한 클라스들의 모임이다. 3장에서 문자렬클라스를 설명할 때 STL의 개념적인 문제를 보여 준다. 그다음에 이에 대하여 구체적으로 설명하였다. 9장에서는 STL의 벡토르용기클라스를 사용하는 방법을 설명하였고 11장과 14장에서는 STL용기클라스의 동작방식과 프로그람적문제를 푸는데서 나서는 알고리듬을 설명하였다.

이 책의 집필에서 고려한 조건

1990년대 전반기부터 새로운 콤퓨터교육과정안이 작성되기 시작하였다. 지금의 과정안과 다른 학교의 과정안을 구체적으로 료해하였다. 료해를 진행한 결과는 다음과 같다.

- 현재의 과정안에서 프로그람작성언어를 취급하지 않았다.
- 겨우 100행정도의 소규모프로그람을 취급하였다.
- 현대적인 설비가 없이 개발하고 있었다.
- 형태를 갖추지 못한 프로그람을 개발하고 있었다. 실세계와의 환경에서 보면 그 차이점을 알수 있다.

콤퓨터전문련습

- 수천수백만개의 행으로 이루어 진 큰 프로그람의 개발을 위해 설계된 프로그람작성언어를 리용한다.
- 그것들을 개발하는것보다 변경, 유지하는 경우가 더 많다.
- 혼자서 작업하지 말고 조를 무어서 작업한다.
- 지시를 받아서 체계를 개발한다.
- 입출력을 위하여 도형사용자대면부(GUI)를 리용하는 체계를 구축한다.
- 체계를 구축하기 위하여 있는 서고들과 도구들을 리용한다.

학생들을 실세계적인 프로그람작성경험을 가진 프로그람수로 준비시키기 위해서 이 책의 첫판이 출판되였다. 3판은 여러가지 의견과 쏘프트웨어공학, 프로그람작성의 두번째 과정안에서 나타난 우결함을 고려하여 다시 출판되였다.

프로그람작성

콤퓨터부문에서 대부분의 중요한 개념들과 문제들은 프로그람이 무엇이며 어떻게 쓰는가를 잘 모르고서는 쉽게 리해할수 없다. 프로그람작성은 좀 어렵다. 프로그람작성은 여러해동안 경험을 쌓아야 한다. 어떻게 교육을 하는가에 따라 조금 달라 질수 있다. 학생들은 잘 작성된 프로그람을 계속 보고 리해하는 과정에 프로그람작성의 묘리를 찾을수 있다. 이러한 단계를 거쳐 학생들은 자기의 의사를 표현하는 방법을 배운다. 학생들은 이러한 능력을 키워서 프로그람을 한개 혹은 여러개의 단락으로 구분하여 작성도 하고 토론도 하며 편집할수 있다.

프로그람작성법을 배워 주는것은 대단히 중요하다. 교제에서는 옳은 프로그람들과 틀린 프로그람들을 많이 보여 주고 설명하였다. 프로그람작성련습은 학생들이 코드를 서술하고 구성할수 있는 능력을 키워 준다. 또한 현재 있는 프로그람을 변경시켜 다른 기능을 수행하도록 할수도 있다. 우리는 학생들이 프로그람을 리해할수 있게 하기 위하여 CD ROM/World Wide Web아이콘을 통하여 이 책에 대한 프로그람들과 내용들을 제공해 준다.

왜 C++를 선택하였는가? 새 과정안을 집행하기 시작하였으므로 어느 프로그람을 선택하여 교육해야 하는가가 제일 중요한 문제로 나섰다. 이전에는 Pascal을 많이 리용해 왔다. Pascal을 교체해야 한다고 결정은 하였지만 어느것을 선정해야 하는지는 결정되지 않았었다. 그때에 사람들이 생각한 대부분의 언어는 C, C++, Modula-3, Schema, Smalltalk 등이였다. 자기의 마음에 드는 언어를 선정하자는 의견들이 제기된 중에서 C, C++를 선택하자는 의견이 제일 많이 제기되였다. 이 제기는 모든 사람들의 동의는 받지 못했지만 앞으로는 객체지향프로그람이 위력한 도구로 될수 있다는 의견을 참작하여 이 언어가 선택되였다. 이 선택은 옳았다. C++는 아주 인기 있고 흥미 있는 프로그람으로 등장하고 있으며 많은 회사들은 이 언어를 자기의 개발도구로 리용하고 있다. 많은 졸업생들이 직업취직을 할 때 C++를 아는가 하는 물음을 자주 받게 된다. C++의 리용분야는 아주 넓다.

JAVA는 GUI를 개발하는 언어로 되고 있지만 C++는 응용프로그람개발언어로서 자기의 능력을 발휘하고 있다. 우수한 콤퓨터수재를 키워 내기 위한 강력한 도구는 C++를 배워 주는것이다. 지난 8년동안 객체지향언어인 C++를 교육한데 의하면 이 프로그람은 초학자들이 능히 리해할수 있는 프로그람이라는것이다.

이전의 과정안에서는 계승, 다중정의, 클라스, 객체 등의 분야를 마지막에 취급하였다. 이렇게 취급한것은 오유이다. 왜냐하면 과정안의 마지막에서 클라스의 새로운 개념을 서술하였기때문에 학생들은 이러한 통합개발환경을 구체적으로 리해하지 못하였다. 또한 책의 마지막에 이러한 내용을 서술하였기때문에 여기에 대해서 파악할 시간도 없었다.

이 책에서부터 표준객체를 취급하기 시작하였다. 3장에서부터 7장까지에 EzWindows API에서 도형처리객체에 대하여 서술하였다. 이러한 객체에 대하여 서술한 다음 7장에서 클라스와 객체를 설계하는 방법을 보여 주었다. 또한 서고, 함수, 구조체를 서술하였다.

쏘프트웨어대상과제

이전에 가르친 내용은 실세계와 아무런 련관도 없는것이였다. 이 책에서는 미래의 콤퓨터과학과 현재 존재하는 체계의 기술적문제들을 해결하는 첫 공정으로서 API를 소개한다. 쏘프트웨어대상과제는 이런것들을 위한 도구이다.

10장과 15장에서 처음 EzWindows API를 취급하였다. API를 리용하여 실세계에서 존재하는 사건구동형프로그람과 도형입출력기능을 리용한 프로그람을 제공할수 있게 하였다.

또한 4개이상의 묶음을 가진 프로그람을 개발할수 있게 하였다. 쏘프트웨어대상과제에서는 쏘프트웨어의 유지에 대하여 설명하였다. 쏘프트웨어대상과제장들의 마지막에 대상과제프로그람의 확장과 변경에 대한 련습을 할수 있게 하였다.

장들에 대한 간단한 해설

제1장. 콤퓨터와 객체지향설계용어

콤퓨터기초용어, 콤퓨터구성, 쏘프트웨어, 쏘프트웨어개발, 쏘프트웨어공학, 객체지향설계와 프로그람작성법

제2장. C++의 기본프로그람구성

함수 Main(), include명령, 설명문, 정의, 프로그람코드서술, 입출력의 호상변환, 기본형, 문자, 상수, 선언, 변수, 변환, 우선권

제3장. 객체의 수정

값주기명령과 변환, 추출, 상수객체, 증가, 감소, 삽입, 추출, 문자렬클라스, STL, 도형처리객체, EzWindows API

제4장. 조종구조

론리값과 연산자, 진리값표 Bool, 관계연산자, 일반적우선권, 전기회로론리값, if명령문, if-else명령문, switch명령문, enum, while명령, for명령, do명령

제5장. 함수사용법과 서고

함수, 파라메터값, 형식파라메터, 실제파라메터, 인용, 조종흐름, 활성화레코드, 모조란수, 원형, 전처리, 포함명령, 머리부파일, 조건부번역, 쏘프트웨어재리용, 서고의 리용, 표준흐름, 처리기, 파일흐름, 파일처리, iostream, iomanip, fstream, ctype, 문자렬, stdlib, 미리 정의된 서고

제6장. 프로그람작성자정의함수

함수정의, 파라메터, 인용, 흐름조종, 귀환명령, 령역, 국부객체, 전역객체, 참조파라메터, 상수파라메터, 기정파라메터, 파라메터계산, 함수의 다중정의, 초기화, 함수이름의 다중화, 우에서부터 아래로의 설계, 재귀, 기억기내부흐름, 함수유효, 표준본보기서고(STL), 2차다항식

제7장. 클라스구축자와 객체지향설계

프로그람작성자에 의한 자료정의, 클라스구축자, 정보의 은폐, 객체지향분석, 설계, 호출지정, 자료성원, 성원함수, 구축자, 만화경프로그람, 객체지향공장자동화모의

제8장. 추상자료형의 실행

자료추상화, 객체지향설계, 기정 및 복사구축자, 검토자, 변이자, 촉진자, 성원값주기, 상수성원함수, 산수연산자다중정의, 참조귀환, 삽입, 추출, 다중정의, 모조란수발생, ADT, 붉은색-노란색-풀색유희

제9장. 목록

1차원배렬, 첨자, 파라메터입력, 초기화, 문자렬, 다차원목록, 표, 행렬, STL, 용기클라스, 적응기클라스, 벡토르클라스, 벡토르성원함수, 분류, 삽입, 고속정렬, 2진탐색, 2차원검색, 목록표현, 초기화목록, 반복자, 로보트에 대한 문제

제10장. EzWindows API의 구체적인 검사

응용프로그람작성자대면부, 도형처리대면부, 사건구동프로그람, 창문, 호출되돌리기, 마우스, 시간사건, EzWindows API기교, ADT를 위한 간단한 창문, 2진화상, 본문표시, Simon says유희

제11장. 지적자와 동적기억기

왼쪽값, 오른쪽값 , 지적자형태, 주소, 간접, 파라메터로서의 지적자, 지적자의 지적자, 상수지적자, 배렬의 평가와 지적자정의, 문자렬처리, 지령행파라메터, 함수지적자, 동적객체, 빈 기억기, new와 delete연산자, 예속지적자, 기억기의 부족, 해체자, 성원값주기, 예약어 this, 옹근수목록 ADT

제12장. 검사와 오유수정

검은 통검사, 흰 통검사, 검사, 단위검사, 통합검사, 체계검사, 명령범위, 값구역, 속박조건, 코드복습, 경로범위

제13장. 계승

객체지향설계, 재리용, 기초클라스, 파생클라스, 단순한 계승, is-a관계, has-a관계, 관계의 리용, 조종계승, 보호성원, 다중계승, ADT 4각형, 원, 타원, 3각형, 객체지향을 리용한 만화경프로그람

제14장. 본보기와 다형성

일반동작과 형, 함수본보기, 클라스본보기, 순서목록, 련결목록, 반복자클라스, Friend, 다형성, 가상함수, 순수한가상함수, 추출기초클라스, 가상파생클라스, 가상다중계승, 목록ADT, 임의접근목록, 순차목록, 목록반복자, 단순련결목록, 2중련결목록

제15장. 쏘프트웨어대상과제-벌레잡이

정보은페, 계승성, 가상함수, 객체지향설계, 벌레잡이유희프로그람, 여러가지 종류의 벌레 ADT, 유희조종자

부록

ASCII문자모임, 일반적우선권표, 입출력흐름, stdlib, 시간, 문자렬, 알고리듬서고, 벡토르, 다른용기클라스, 문자렬클라스, 이름공간, 명령문리용, 례외, friend, EzWindows API, 대상과제파일과 제작파일.

기호

이 책에는 다음과 같은 그림기호들이 있다.

이 기호는 프로그람작성에서 있을수 있는 주의점을 나타낸다. 일반적인 프로그람오유가 어떻게 나타나는가를 설명한다.

이 기호는 프로그람작성형식과 관련된 대상을 가리킨다. 관계의 수가 포함되며 이 책에서 코드가 표시하는것은 위력한 관계를 반영한다.

이 기호는 C++프로그람작성언어와 관련한 내용들을 가리킨다. 이 기호에는 두가지 형태가 있는데 한가지 형태는 개발된 C++를 위한 기호이고 다른 하나는 쏘프트웨어개발에서 새로운 언어의 확장을 서술한다.

이 기호는 프로그람작성에서 알아야 할 문제들을 구체적으로 설명한다.

이 기호는 콤퓨터의 력사를 설명하는데 리용된다. 많은 사람들은 프로그람을 간단히 쓰는것이 콤퓨터작업이라는것을 잊어 버리군 한다. 프로그람설계와 작성은 콤퓨터작업의 중요한 부분이다. 매장에서 적어도 한번씩 콤퓨터의 유래와 발전에 대하여 소개한다.

아래에 C++를 학습하는데서 필요한 참고서를 소개한다.

International Standard for Information Systems-Programming Language C++, ISO/IEC FDIS 14882, Washington, DC: American National Standards Institute, 1998.

B.Stroustrup, The C++ Programming Language, 3rd ed., Reading, MA: Addison-Wesley, 1998.

The following are good sources on libraries and more-advanced object-oriented design, program developmemt, and the Standard Template Library.

J.Bergin, Data Abstraction: The Object-Oriented Approach Using C++, New York: McGraw-Hill, 1994.

M.D.Carroll and M.A.Ellis, Designing and Coding Reusable C++, Wesley, 1995.

A.Doenig and B.Moo.Ruminations on C++, Reading, MA: Addison, Wesley, 1997.

S.B.Lippman and J.Lajoie, C++ Primer, 3rd ed, Reading, MA: Addison-Wesley, 1998,

S.Maguire, Writing Solid Code, Redmone, WA: Microsoft Press, 1993.

S.Meyers, Effective C++, Reading , MA:Addison-Wesley, 1998.

S.Meyers, More Effective C++, Reading, MA:Addison-Wesley, 1996.

D.R.Musser and A.Saini, STL Tutorial and Reference Guide, Reading, MA:Addison-Wesley, 1995.

P.J.Plauger, A.Stepanov, M.Lee, and D.R.Musser, The Standard Template Library,Englewood Cliffs, NJ:Prentice-Hall, 1998.

B.Stroustrup.The Design and Evolution of C++ , Reading, MA:Addison-Wesley, 1994.

아래에 콤퓨터의 력사를 학습하는데서 필요한 참고서를 소개한다.

S.Augarten, Bit by Bit: Art Illustrated History of Computers . New York: Ticknor & Fields, 1984.

SP.J.Dening and B.Metcalfe(eds.). Beyond Calculation: The Next Fifty Years of Computing, New York: Copernicus Press, Springer-Verlag, 1997.

J.A.N.Lee, Computer Pioneers, Piscattaway, NJ:IEEE Press, 1995.

J.Palfreman and D.Swade, The Dream Machine: Exploring the Computer Age, London:BBC Books, 1991.

H.G.Stine, The Untold Story of the Computer Revolution, New York: Arbor House, 1985.

M.R.Williams, A History of Computing Technology, Englewood Cliffs, NJ:Prentice-Hall, 1985.

제 1 장. 콤퓨터와 객체지향설계의 용어

소개

콤퓨터는 새 세기 우리 생활에서 중요한 자리를 차지한다. 지금 우리는 인터네트세계에서 살고 있다. 콤퓨터는 여러 부문에서 널리 리용되고 있다. 사람들이 흔히 사용하는 전화도 콤퓨터로 조종할수 있으며 앞으로 비행기는 조종사가 없이 콤퓨터체계에 의하여 리착륙할수 있으며 날아 갈수 있다. 콤퓨터체계는 2개의 기본구성부분인 하드웨어와 쏘프트웨어로 되여 있다. 하드웨어는 콤퓨터장치를 의미한다.

쏘프트웨어는 콤퓨터에 명령을 주는 프로그람이다. 우에서 말한 전화체계에서 호출대기와 같은 특수한 기능을 제공하는것이 쏘프트웨어이다. 쏘프트웨어의 설계와 작성방법은 현재 수백가지나 된다.

객체지향프로그람의 설계와 작성은 좀 복잡한 반면에 그 관리와 복제에서 믿음성이 있다. 이 장에서는 객체지향설계와 관련되는 기본용어와 개념을 소개한다.

기본개념

- CPU
- 2진수체계
- 기계어/체계쏘프트웨어
- 응용쏘프트웨어
- 정보은폐
- 모듈화
- 체계
- 재리용
- 조작체계
- 번역체계
- 콤파일러
- 추상화
- 객체지향설계
- 객체지향언어
- 계승
- 다형성

1.1 콤퓨터기초용어

새 학문을 배우는데서 중요한것은 그 용어에 정통하는것이다. 콤퓨터학문인 경우에 콤퓨터학자들이 콤퓨터로 진행하는 모든 과정에 대하여 첫 글자를 모아 만든 단어와 략어를 사용하기 좋아하므로 더욱 그러하다. 용어를 모른다면 두 콤퓨터전문가사이의 대화를 리해할수 없다. 콤퓨터가 실로 많은 용어를 가지고 있기때문에 이 용어를 쓰지 않고 콤퓨터에 대하여 론할수 없다.

1.1.1 측정단위

콤퓨터전문가들이 사용하는 많은 콤퓨터용어에는 콤퓨터와 관련된 측정량에 대한 용어도 있다. 이 량들에는 장치의 크기나 용량, 속도를 표시하는것들이 있다. 속도면에서 콤퓨터전문가들은 어떤 조작을 하는데 시간이 얼마나 걸리는가를 자주 론의한다. 여기서 측정단위는 백만분의 1s 또는 10억분의 1s이다. 흔히 사용하는 량들을 표 1-1에 보여 주었다. 여기서 보는바와 같이 지금은 콤퓨터가 나노초(ns)를 리용하지만 앞으로는 피코초(ps)를 단위로 리용하게 될것이다. 콤퓨터전문가들은 또한 속도로서 연산시간대신 박자주파수를 리용하기도 한다.

박자주파수는 1초동안에 콤퓨터가 수행하는 연산단위인데 초당 싸이클 즉 Hz(헤르쯔)로 표시한다.

실례를 들어 박자주파수가 5억Hz인 콤퓨터는 2ns동안에 한개 동작을 수행한다.

표 1-1. 콤퓨터속도측정의 일반단위

10의 제곱	값	단 위
10^{-3}	1/1,000	㎳(미리초)
10^{-6}	1/1,000,000	㎲(마이크로초)
10^{-9}	1/1,000,000,000	㎱(나노초)
10^{-12}	1/1,000,000,000,000	㎰(피코초)

표 1-2에 측정체계에서 많이 쓰이는 량을 표시하는 앞붙이를 주었다. 앞에서 실례를 든 5억Hz라는 표현을 500MHz라고 바꾸어 표현할수도 있다.

표 1-2. 자주 사용되는 10의 제곱을 표시하는 앞붙이

값	단 위
1,000	K(Kilo)
1,000,000	M(mega)
1,000,000,000	G(giga)
1,000,000,000,000	T(tera)

콤퓨터전문가들은 용량이나 크기의 측정에서 2의 제곱을 자주 리용한다. 왜냐하면 콤퓨터가 2진수체계로 동작하기때문이다. 자주 쓰이는 2의 제곱을 표 1-3에 주었다. 표에서 보는바와 같이 2의 10제곱의 략어는 K(키로)인데 K라는것은 "kilo"에서 나온 문자이다. 1024는 1000근방이므로 앞붙이 K(키로)로 대신한다. 또한 2^{20}은 1000000근방이므로 앞붙이 M(메가)로 대신한다. 이러한 단위들은 기억용량을 표시하는데 리용된다. 그러나 실지로는 2의 정확한 제곱값으로 계산해 보아야 한다. 례를 들어 128Mbyte는 128×2^{20}byte 즉 134,217,728byte이다.

여러가지 의미를 가지는 이러한 앞붙이들의 사용은 때때로 혼란을 가져 올수도 있다. 그러나 문맥을 따져 보면 그 의미를 정확히 가를수 있다. 만일 어떤 사람에게 30K$의 돈을 준다고 말한다면 그것은 30,720$(즉 30×1024)가 아닌 30,000$를 의미한다.

표 1-3. 자주 리용되는 2의 제곱들과 앞붙이

2의 제곱	값	앞붙이
2^{10}	1,024	K(kilo)
2^{20}	1,048,576	M(mega)
2^{30}	1,073,741,824	G(giga)
2^{40}	1,099,511,627,776	T(tera)

1.1.2 콤퓨터의 구성

콤퓨터는 4개의 부분으로 구성되여 있다(그림 1-1). 콤퓨터의 기본핵심은 중앙처리장치(CPU)인데 이 장치는 기본연산과 조종을 맡아 수행한다. 기억기는 자료와 프로그람을 기억해 두는 장소이다. 그림 1-1에서 화살표는 CPU가 기억기에서 정보를 불러 내고 다시 써넣을수 있다는것을 의미한다. 사람과 콤퓨터사이에 정보를 주고 받아야 하므로 입력장치와 출력장치도 중요한 구성부분으로 된다. 콤퓨터의 4가지 구성부분에 대하여 보기로 하자.

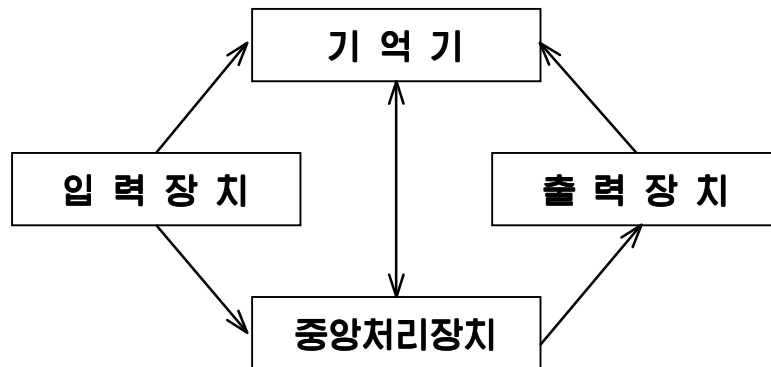



그림 1-1. 콤퓨터의 구성

CPU는 산수/론리연산을 진행하는 장치이다. CPU의 산수/론리장치(ALU)는 +, ×, -, /와 같은 산수연산을 진행한다.

콤퓨터는 수를 표시하기 위하여 2진수체계를 리용한다. 2진수체계는 0과 1 값을 가진다. 콤퓨터에서 2진수체계는 전기회로에서 On/Off스위치와 류사하다. 스위치의 절환상태는 수값을 나타낸다. 이전의 콤퓨터들에서 이 스위치들은 기계식으로 되여 있었다. 그러므로 장치가 무거웠으며 부피가 크고 많은 전력이 소비되였다.

지금의 콤퓨터에서 이 스위치들은 매우 작은 반도체3극소자로 되여 있다. 그리하여 콤퓨터의 CPU를 하나의 규소반도체소편으로 실현할수 있다. CPU가 한 소편으로 되여 있기때문에 극소형처리소편이라고도 한다.

그림 1-2. PentiumⅡ처리소편

인텔의 PentiumⅡ처리소편은 가로 4.8inch, 세로 6.0inch인 소편에 거의 7500,000개의 3극소자를 가지고 있다(그림 1-2). PentiumⅡ, PentiumⅢ의 여러가지 소편들은 거의 28,000,000개의 3극소자를 가지고 있다.

10진수체계는 흔히 쓰는 수체계이다. 여기서 수자의 위치에 따라 해당한 값을 가진다.

실례로 10진수 4506에서 5는 100의 자리이므로 500을 의미한다. 오른쪽에서부터 수를 읽어서 매 수자를 10의 증가제곱으로 표시한다. 따라서 4506은 $4\times10^3 +5\times10^2 +0\times10^1 +6\times10^0$ 으로 표시할수 있다.

2진수체계는 2의 증가제곱을 리용한것을 내놓고는 10진수체계와 같다. 실례로 2진수 1101은 10진수 13(8+4+0+1)을 $1\times2^3+1\times2^2+0\times2^1+1\times2^0$ 으로 표시할수 있다.

10진수가 아닌 다른 수를 표시하는데서도 그 수의 밑수를 표시하여 모든 수를 나타낼수 있다. 그러므로 $(100100)_2$는 10진수 36을 2진수형태로 표시한것이며 $(1001)_2$은 10진수로 9이다. 2진수의 개별적인 수자를 비트라고 한다.

아주 큰 2진수를 표시하기가 힘드므로 2진수의 비트값은 흔히 더 큰 진수로 묶어서 표시할수 있다(물론 2의 제곱이다). 오른쪽으로부터 시작하여 세개의 수를 묶어서 8진수형태로 표시할수 있다($2^3=8$). 수 $(01011101)_2$는 오른쪽으로부터 시작하여 3개씩 묶어서 8진수형식으로 변환할수 있다. 구체적으로 마당 $(01)(011)(101)_2$로 가르고 개별적인 묶음을 8진수로 변환할수 있다. 매 수마당을 변환하면 수 $(135)_8$로 된다.

$(01)_2=1$, $(011)_2=3$, $(101)_2=5$

이 값은 2진수를 10진수로 표시한 값이다. 차이점은 8의 제곱을 사용하는것이다. 수 $(135)_8$은 $1\times8^2+3\times8^1+5\times8^0$ 인데 10진수 93과 같다. 2진수나 8진수형식의 수를 10진수의 수로 변화하는것은 그리 어렵지 않다.

10진수체계를 8진수체계로 변환하기 위하여(같은 값을 2진수로 변환하는것보다 쉽다.) 8진수를 ...WXYZ로 표시해 보자. 그러면

$W\times8^3+X\times8^2+Y\times8^1+Z\times8^0$ 혹은 ... $W\times512+X\times64+Y\times8+Z\times1$

로 표시된다. 여기서 첫 단계는 수에 대해서 8의 배수를 설정하는것이다. 이 값은 8로 나누어 계산할수 있다. 여기서 나머지값은 Z이다. Y값은 앞의 연산에서 나온 상을 다시 8로 나누어 계산할수 있는데 그 값은 64의 배수로 된다. 나머지는 Y값이다. 이러한 처리는 나누어 지는 수가 8보다 작아 질 때까지 계속한다. 10진수 458을 8진수와 2진수로 변환하는 실례를 구체적으로 보자. 계산은 다음과 같다.

458/8=57, 나머지 2
57/8=7, 나머지 1
7/8=0, 나머지 7

그러므로 $(458)_{10}$은 8진수 712와 같다. 2진수형식은 매 8진수를 확장시키면 쉽게 얻을수 있다. 즉

┌7 ┌1 ┌ 2
111 001 010

으로 되며 2진수 111001010이 주어 진다.

대부분의 콤퓨터에서 기억의 표준단위는 8bit이다. 1Byte는 8bit이다. 8진수체계는 3개의 수를 묶으며 8로 표시하기 힘들 때 16진수가 리용된다. 16진수체계에서는 매 비트들을 4개씩 묶는다. 따라서 1byte에는 2개의 16진수가 있다. $(89)_{10}$을 2진수형식으로 주고 (101) (1001)오른쪽마당으로부터 시작하여 4개씩 묶어 나가면 $(59)_{16}$은

$5\times16^1 + 9\times16^0 = 80+9 = 89$

로 된다. 수를 정확히 변환하였는가를 검사하기 위하여 10진수로 다시 변환할수 있다. 2진수나 8진수를 10진수로 변환하기 위해서는 우와 같은 방법을 리용한다. 그리고 16진수인 경우에는 16개의 가능한 값을 주어야 하므로 보충적기호로서 9보다 더 큰 수들을 표시하여야 한다. 이때 수 10부터 15까지를 표시하기

위하여 문자 A로부터 F까지를 사용한다. 따라서 8진수 712는 16진수로 1CA이며 매 8진수의 2진수등가값을 써서 얻을수 있고 다음의 도식에서 보여 주는것처럼 오른쪽으로부터 시작하여 4개씩 비트렬을 가른다.

```
┏1        ┏12=C        ┏10=A
1         1100         1010
```

2진수로 연산을 진행하는것은 10진수로 하는것과 같다. 더하기와 곱하기를 둘 다 그림 1-3에 보여주었다. 2진수는 오른쪽으로부터 시작하여 더하며 두수의 합이 1보다 크면 자리올림을 그 앞비트에 등록한다. 곱하기에서 곱해 질 수(짝대기수)는 오른쪽으로부터 시작하여 곱하는 수에 의해 곱해 진다. 2진수체계에서는 항상 0, 1을 곱하므로 곱하기가 아주 간단하다. 모든 부분들을 구하여 마지막단계에서는 최종결과를 얻는다.

0과 1로 구성된 체계에서는 《+》나 《-》기호를 위한 공간이 없으므로 부의 옹근수값을 표시하는데서 여러가지 약속을 하여야 한다.

모든 콤퓨터들은 값을 표시하기 위하여 2진수체계를 리용한다. 기억의 기본단위는 단어(word)이다. 콤퓨터는 자료를 8bit단위로 처리한다. 이 8bit를 1byte라고 한다. 이미 알고 있지만 정의 옹근수 0부터 255(2^8 -1)까지를 이 단위로 나타 낼수 있다. 그런데 어떤 수가 부수로 표시(2진수의 맨 왼쪽수자가 1로 표시된 경우)되거나 덜기연산을 진행하려면 맨 왼쪽의 수자 1(부호비트라고 한다.)을 무시해야 한다.

```
2진수 더하기      10진수 더하기
   00010              2
 + 00111            + 7
 -------            ---
   01001              9

2진수 곱하기      10진수 곱하기
    0101              5
  × 0011            × 3
  ------            ---
   00101             15
  00101
 00000
+ 00000
--------
 00001111
```

그림 1-3. 2진수더하기와 곱하기

대부분의 콤퓨터에서는 옹근수를 표시하기 위하여 2의 보수체계를 사용한다. 2의 보수체계에서 정수와 령은 앞에서 본것처럼 표시한다. 그런데 부수는 여러가지로 표시할수 있다. nbit를 단위로 하였을 때 -N의 2의 보수에 의한 표시는 2진수로 2^n-N이다.

구체적으로 실례를 들어 보자. 다시 8bit단위로 작업한다고 가정하자. 3에 대한 2의 보수표시는 00000011이다. -3의 2의 보수표시는 2^8 -3 혹은 253으로서 2진수값이다. 253의 2진수표시는 11111101이다. 따라서 -3의 2의 보수표시는 11111101이다. 부의 옹근수에 대한 2의 보수를 얻기 위한 가장 쉬운 방법을 아래에 보여 준다.

1단계 : 옹근수를 2진수로 표시한다.
2단계 : 매 bit를 반전시킨다(1은 0으로, 0은 1로).
3단계 : 보수에 1을 더한다 .

-127_{10}값을 실례로 들어 보자.

1단계 : 01111111(127_{10}은 8개의 2진수범위내에 있다.)
2단계 : 10000000(반전한다.)
3단계 : 10000001(반전한 비트에 1을 더한다.)

따라서 -127_{10}의 2의 보수는 10000001이다. 정확히 계산되였는가 알아 보려면 1단계값과 3단계값을

더하여 얻어 진 수가 2단계값보다 0마당이 하나 더 늘어 났는가를 따져 보면 된다. 즉

$$
\begin{array}{r} 01111111 \\ +10000001 \\ \hline 100000000 \end{array}
$$

이다.

2의 보수체계를 사용하여 표시된 수를 가지고 2진연산을 진행할 때 대부분의 비트의 값은 떼여 버린다. 따라서 합은 0이다. 우의 실례에서 2의 보수체계단위의 가장 높은 비트는 부호비트로서 사용된다. 가장 높은 비트위치의 1은 부수임을 지정하며 0은 정수임을 지정한다. 8bit를 단위로 표시할수 있는 값범위는 -128부터 127까지이다. 일반적으로 nbit렬을 단위로 하여 2의 보수법으로 표시할수 있는 값범위는 $-2^{n}-1$부터 $2^{n-1}-1$까지이다.

이 보수는 또한 별도의 수단을 사용하지 않고 덜기를 진행하는 방법에 의하여 콤퓨터를 동작시킨다. 두 2진수 x와 y의 차를 계산하기 위하여 y에 대한 2의 보수를 구하고 그다음 x에 더한다. 결과는 차 x-y이다.

CPU의 두가지 중요한 특징은 조종할수 있는 수의 크기와 산수연산을 얼마나 빨리 수행할수 있는가 하는것이다. CPU가 조종할수 있는 수의 크기는 보통 콤퓨터가 다루는 가장 큰 옹근수의 비트렬의 길이만큼 주어 진다. 2∼3년전에 일반적으로 기계가 다룰수 있는 가장 큰 옹근수는 32bit였다. 지금 CPU들은 64bit옹근수들을 다룰수 있게 되였다. nbit옹근수들은 $0\sim2^{n-1}$까지의 10진수를 표시할수 있다. 실례로 8bit옹근수로 동작하는 콤퓨터에서 가장 큰 옹근수는 $2^{8}-1$ 혹은 255이다.

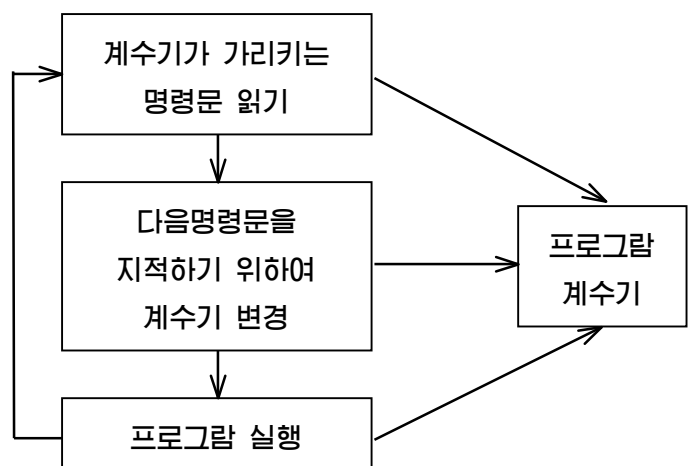
CPU의 산수연산속도는 흔히 2개의 옹근수를 더하는데 걸리는 시간에 의하여 정의된다. 1940년대에 만든 콤퓨터들은 150∼200μs동안에 2개의 32bit수들을 더할수 있었다. 오늘 현대적인 콤퓨터는 1ns동안에 2개의 32bit수들을 더할수 있다.

CPU는 산수론리연산장치(ALU)외에 조종장치도 포함한다. 조종장치는 기억기로부터 명령을 불러 내고 그 명령에 의해 일정한 동작을 수행한다.

명령에 의하여 동작을 수행하는것을 명령실행이라고 한다. 기억기에 있는 명령이란 프로그람을 말한다. 조종장치는 명령을 읽어 내고 실행하는 이러한 단계들을 차례차례로 실행한다. 이 처리를 명령읽기/실행주기라고 한다. 그 실례를 그림 1-4에 보여 준다.

명령읽기와 명령실행을 추적하기 위하여 조종장치는 또한 프로그람계수기를 리용한다. 프로그람계수기는 읽고 실행하는 다음명령의 기억기주소를 가리킨다. 대부분의 프로그람계수기는 프로그람을 련속적으로 갈라 준다. 즉 기억기에 보관된 명령을 읽어 내고 실행한다. 그런데 명령을 련속적으로 실행할수 있다면 콤퓨터사용은 제기되는것이 없다. 조종장치는 기억기로부터 읽어 낸 명령의 자료바이트수를 검사하고 검사결과에 기초하여 프로그람계수기를 변화시킨다.

그림 1-4. 콤퓨터의 읽기/실행주기

이러한 기능은 CPU가 명령실행에 기초하여 다음 동작을 결정하게 된다. 이 결정은 모

든 계산장치들에 대하여 기본핵으로 된다. CPU는 주기억기와 련결되여 있는데 기억기에는 CPU에서 실행하는 명령과 자료가 들어 있다.

앞에서 언급한것처럼 현대적인 콤퓨터들에서 주기억기는 8bit 즉 1byte를 단위로 하여 정보들을 기억시켜 놓고 있다. 주기억기의 중요한 특성은 정해 진 시간에 임의의 주소나 바이트를 호출할수 있다는것이다. 그러므로 주기억기를 흔히 자유호출기억기 즉 RAM이라고 한다.

정보가 보관된 테프에서는 정보를 읽기 위하여 정확한 위치로 테프를 전진시켜야 한다. 이 부분에서 정보를 호출하는 시간은 정보의 위치에 관계된다. 이런 형태의 기억기를 순차호출기억기라고 한다.

주기억기의 두가지 중요한 지표는 그것의 용량과 속도이다. 주기억의 용량은 바이트단위로 표시한다. 개인용콤퓨터에서 주기억기는 64Mbyte부터 256Mbyte까지이다. 흔히 128Mbyte의 기억기를 가진 콤퓨터가 많다. 주기억기의 속도는 지정된 위치로부터 정보를 읽어 내는데 걸리는 시간을 의미한다. 개인용콤퓨터에서 대표적인 속도는 10～60ns사이이다. RAM은 콤퓨터의 전원을 끌 때 그안의 내용을 잃어 버린다. 이런 형태의 기억기를 휘발성기억기라고 한다.

전원을 꺼도 내용을 그대로 보존하는 기억기들도 있다. 대부분의 콤퓨터들은 RAM외에 읽기전용기억기 즉 ROM을 더 가지고 있다. 이런 형태의 기억기는 비휘발성기억기이다. 즉 보관된 정보는 콤퓨터를 껐을 때에도 그대로 남아 있게 된다. ROM의 내용은 콤퓨터제작자들이 설치한다. 일단 설치되면 ROM은 다시 써넣기할수 없으며 읽기만 할수 있다. 때문에 읽기전용기억기라고 한다.

이 기억기를 보통 ROMBIOS(ROM기본입출력체계)라고 하는데 콤퓨터를 처음 켰을 때 기동하는 콤퓨터의 기종과 명령을 판단하는 정보를 가지고 있다. 콤퓨터를 켜고 ROMBIOS프로그람이 실행될 때의 처리를 초기적재라고 한다. CPU는 계산을 진행하고 주기억에 프로그람의 결과를 보관하였다가 그 결과를 다른 기억기(보조기억기)에 돌려 준다. 그리하여 콤퓨터를 꺼도 이미 작성하였던 결과를 잃어 버리지 않고 그 정보를 보관하여 둘수 있다.

입력장치와 출력장치는 두가지 기능을 수행한다. 콤퓨터에 명령과 자료를 넣기 위한 입력장치는 현재 수십가지나 된다. 가장 대표적이면서 일반적인 장치는 건반, 마우스, 그리고 CD-ROM구동기이다. 흔히 쓰이지 않는 장치들로서 스캐너(scanner), 음성입력장치, 조종간, 빛펜들이 있다. 또한 많은 출력장치들이 있는데 정보를 서로 다른 형식으로 가공하여 출력할수 있다. 일반적인 출력장치에는 현시장치와 레이자인쇄기, 잉크분사식인쇄기, 작도기 그리고 스피카 등이 있다.

일반적이면서 대표적인 입출력장치들을 그림 1-5에서 보여 주었다.

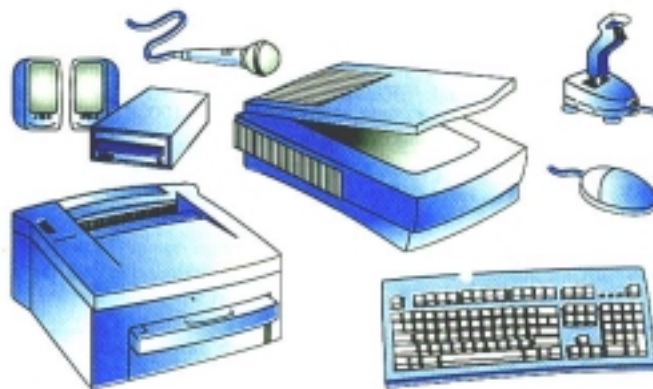



그림 1-5. 대표적인 입출력장치들

일부 장치들은 입력과 출력을 다 조종할수 있으며 입출력장치로 사용된다. 대부분의 입출력장치들은

자기기록기술에 의하여 동작을 진행한다. 대표적인 입출력장치는 플로피디스크구동기, 하드디스크구동기, 자기테프장치들이다. 플로피디스크구동기는 적은 용량의 기억기에 정보를 쓰고 읽어 낼수 있다. 처음에 플로피디스크구동기는 360Kbyte의 정보만을 기록할수 있었다. 지금 플로피디스크구동기는 1.44Mbyte의 용량을 가지고 있다. 많은 장치들은 Zip구동기를 포함하고 있다. 이 분리가능한 디스크는 100 ~ 250Mbyte의 정보를 기록할수 있다. 하드디스크는 플로피디스크나 Zip원판과는 달리 아주 큰 용량을 가지고 있다. 하드디스크에는 대체적으로 30～40Gbyte의 정보를 담을수 있으며 75Gbyte이상의 정보를 담을수 있는 하드디스크가 개발되여 리용되고 있다.

하드디스크는 정보의 읽기쓰기를 방해하는 먼지와 다른 물리적요인을 막기 위하여 밀페된 통안에 넣었다. 하드디스크를 사용하기전에 초기화를 하여야 한다. 자료를 효과적으로 쓰고 읽어 내기 위하여 초기화조작이 필요하다. 디스크초기화는 주어 진 크기와 자리길수에 맞게 자리길을 그리는 처리를 상사적으로 진행한다. 이런 과정을 거치게 되면 정보를 효과적으로 쓰고 읽어 낼수 있다. 초기화를 한후 디스크의 공간이 정리되는 리유는 초기화에서 자리길(track)과 분구(sector)번호를 달기때문이다.

개인용콤퓨터에서 매우 중요한 출력장치는 현시장치이다. 현시장치는 흔히 CRT 혹은 음극선관으로 되여 있는데 텔레비죤수상기의 수상관과 같다. 현시장치는 도형기판이라는 출력장치에 의하여 조종된다. 도형기판은 수상관에서 처리할수 있는 형태로 자료를 변환하여 보낸다. 수상관과 도형기판의 중요한 특징은 재생속도, 해상도, 색수이다. 재생속도는 도형기판이 화면에 영상을 얼마나 빨리 그리는가 하는것이다. 이러한 공정은 수상관에서 사용하는 밝기조종회로에 의하여 밝아 지거나 어두워 지면서 주기적으로 진행된다. 60kHz와 같은 낮은 재생속도는 영상이 미세하게 떨므로 눈에 피로를 줄수 있다. 눈으로 화면을 보면 깜박이는것을 인차 볼수 있다. 많은 도형기판은 70～100kHz까지의 속도로 화면을 재생할수 있다. 이 정도의 속도이면 화면의 움직임에 의한 눈의 피로를 없앨수 있다.

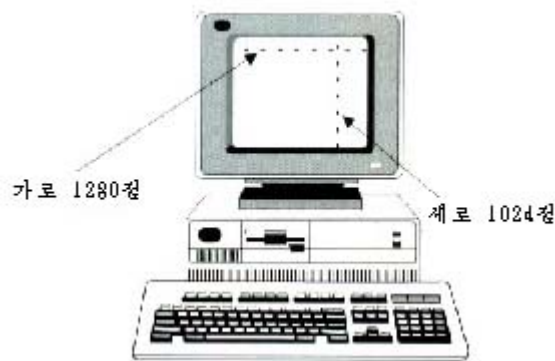



그림 1-6. 탁상형콤퓨터에서 현시장치 즉 수상관

평판형현시장치는 CRT와는 다른 형태의 출력장치이다. 이 현시장치는 CRT보다 여러가지 우점을 가지고 있다.

첫째로 CRT를 사용하지 않으므로 부피가 훨씬 작다. 콤퓨터에서 평판형현시장치를 리용하면 콤퓨터전체의 크기를 작게 만들수 있다.

둘째로 평판형은 CRT를 사용하지 않기때문에 에네르기소비가 적고 전자선이 나오지 않는다.

셋째로 평판형은 말그대로 평판이다. 따라서 화상이 이지러 지지 않는다. 실지 평판형현시장치는 같은 크기의 CRT현시장치보다 값이 비싸다. 그러나 가격은 내려 가기 시작하였다.

도형기판의 다른 특성은 해상도와 색수이다. 이 두가지 특성은 서로 련관되여 있다. 해상도는 화면

의 가로와 세로의 화소수를 인치당으로 나타낸다. SVGA에 의하여 제공되는 표준해상도는 800×600이다. 즉 화면의 가로를 800화소, 세로를 600화소로 주사한다. 도형기판은 기억기에 매 화소의 정보를 보관한다. 이 방법은 CPU의 연산과 병행하여 항상 화면에 자료를 현시하게 할수 있다. CPU와 도형기판은 정보가 변화될 때에만 통신을 진행해야 한다. 더 높은 해상도를 가진 도형기판은 보다 많은 기억기를 요구한다. 실례로 많은 도형기판들은 1024×768～1920×1440까지의 해상도를 제공하고 있다. 이러한 기판들은 16～32Mbtye정도의 기억기를 요구한다. 기억기용량과 해상도에 관계되는것은 표시되는 색수이다. 256색(28)을 현시하기 위하여서는 매 화소에 대해서 1byte가 필요하다. 《자연색》에는 1670만개의 색이 있는데 한 화소당 24bit가 요구된다. 따라서 현시할수 있는 색의 수와 해상도는 둘 다 도형기판에서 기억기의 량에 관계된다는것을 알수 있다. 기억기가 크면 클수록 해상도와 색수가 높아 진다는것을 의미한다. 대부분의 도형기판에서는 해상도가 낮을수록 더 많은 색수를 표시할수 있다. 실례로 800×600해상도에서는 《자연색》을 현시할수 있지만 1024×768과 같은 더 높은 해상도에서는 65536까지의 색밖에는 표시하지 못한다.

가장 일반적인 두가지의 입력장치는 건반과 마우스이다. 건반에서 건을 누르면 쏘프트웨어는 눌린 건을 읽고 적당한 코드로 변환한다. 마우스는 매우 편리한 입력장치이다. 일반적으로 마우스는 한두개의 단추와 밑에 있는 자그마한 공을 가지고 자료를 입력할수 있다. 마우스를 굴리면 화면의 지시자가 따라 움직인다. 화면상의 지시자는 보통 화살표처럼 표시된다. 화살표가 화면의 지정된 지역을 지정하도록 마우스를 움직이고(실례로 응용프로그람에 의해 현시되는 차림표) 마우스단추를 찰칵하여 콤퓨터가 차림표 항목에서 지정된 명령을 수행하도록 할수 있다. 콤퓨터에 명령을 주는 차림표와 단추를 사용하는것은 아주 편리하게 되여 있다.

문제

1. 1/1012를 계산하시오.
2. 1/1015를 계산하시오. 이에 대한 대답이 없다.
3. 디스크가 30Gbyte의 기억공간을 가진다. 디스크는 정확히 몇byte인가?
4. CPU란 무엇인가?
5. 10진수 38을 2진수로 변환하시오.
6. 2진수 010101을 10진수로 변환하시오
7. 10진수 555는 8진수로 얼마인가?
8. 10진수 4256은 16진수로 얼마인가?
9. 옹근수 -101의 2의 보수는 얼마인가?
10. 읽어 들여 실행하여야 할 다음명령을 무엇이 지정하는가?
11. RAM의 의미는 무엇인가?

1.1.3 프로그람작성

콤퓨터는 혼자서 아무 작업도 할수 없다. 여러가지 지정된 과제를 수행하기 위해서는 콤퓨터에 지시를 주는 프로그람이 있어야 한다. 여러가지 과제를 수행하도록 프로그람을 작성하면 콤퓨터의 능률을 높일수 있다. 프로그람은 콤퓨터가 해야 할것을 알려 주는 명령의 묶음이다. 명령은 콤퓨터에 명령을 주기 위하여 설계된 언어로 작성한다. 이러한 언어를 프로그람작성언어라고 한다.

프로그람작성언어의 한 형태는 기계어이다. 기계어프로그람은 콤퓨터가 직접 리해할수 있는 프로그

람이다. 기계어는 콤퓨터가 수행하는 기본연산을 말아 수행하는 명령들로 구성되여 있다. 따라서 콤퓨터의 기종에 따라 서로 다른 기계어들을 사용한다. 실례로 인텔의 Pentium에서 리용하는 기계어는 IBM의 POWER PC에서 리용하는 기계어와는 전혀 다르다. 콤퓨터설계에서 기본부분은 콤퓨터가 수행할수 있는 기본조작과 명령을 기계어로 코드화하는것이다. 기계어명령을 2진수로 코드화하는것은 그 명령에 대한 비트렬을 만드는 과정이다. 대부분의 장치들은 +, -, ×, /와 같은 연산조작을 수행하는 명령들을 가지고 있다. 명령의 또 다른 형태는 뛰여넘기명령인데 이 명령은 명령주소계수기를 변경시킬수 있다.

오늘날의 현대적인 콤퓨터에서 원시적인 기계어로 직접 프로그람을 작성하는 일은 없다 만일 작성하자면 몹시 품이 많이 든다. 여기에 대해서 작은 프로그람인 《Pop Machine 100》 즉 PM 100을 실례로 들어 알아 보자. PM100은 사이다판매기에서 리용하기 위해서 설계되였다. 명령묶음은 매우 간단하다. PM에서 기계어명령은 7bit로 구성되여 있다. 처음의 3bit는 수행해야 할 동작을 지정한다. 이 비트들은 조작코드 혹은 OP코드로 표시한다. 나머지 4bit는 뛰여넘기명령에 의해서만 리용된다. 이 4bit들은 다음명령을 호출하기 위한 기억기주소를 나타낸다. PM100은 기억기위치 16에 들어 있을수 있다. 기계의 명령묶음을 표 1-4에 주었다.

뛰여넘기명령을 분석해 보자. 부호화는 110 AAAA이다. 이 기계어명령의 두 부분사이의 공백은 주소부분과 명령코드부분을 구분하기 위한것이다. 이렇게 구분하면 기계어명령을 편리하게 수행할수 있게한다. 앞부분의 명령코드인 비트패턴 110은 뛰여넘기연산을 수행한다. 뒤의 주소를 가리키는 부분은 4개의 비트 AAAA로 표시되여 있다. PM100이 뛰여넘기연산을 수행할 때 주소를 가리키는 4개의 비트는 프로그람계수기에 의하여 관리된다(그림 1-4). 돈을 넣은 다음의 지령은 다음과 같이 수행되도록 해야한다. 많은 돈을 넣었을 때 사이다 한통을 내주고 남은 돈에 따라 동작하도록 PM100기계어프로그람을 작성해야 한다. 사람들이 기계에 위조돈을 넣을수 있는 우려가 있으므로 위조돈을 발견하면 프로그람은 PM100이 사진을 경찰서에 보내면서 경찰을 부르도록 한다. 그다음 재설정한다. 혹시 범죄자가 당황하여 경찰이 체포하려 오는데 오랜 시간이 걸리게 하기 위하여 기계를 파괴할수도 있다.

표 1-4. PM100 명령

명령해설	2진수 부호화
기계를 재설정한다	000 0000
값을 기다린다	001 0000
화폐가 위조품이 아니면 다음의 명령에로 넘어 간다	010 0000
총합에 값을 더한다	011 0000
총합보다 그 값이 작으면 다음의 명령에로 넘어 간다	100 0000
위조지페이면 사진을 주고 경찰을 부른다	101 0000
기입위치로 이동한다	110 AAAA
사이다통을 내준다	111 0000

그렇지만 경찰은 범죄자의 사진을 가지고 있다. PM100은 위조돈을 발견하기 위한 특수명령을 가지고 있다. 또한 경찰에게 전화하고 기계앞에 서 있는 사람을 찍은 사진이 들어 있는 사진기를 돌리는 명령도 있다. 다음에 요구하는 동작을 수행하기 위한 기계어프로그람을 주었다.

기억기위치	명령	해설
0000	000 0000	기계를 재설정한다
0001	001 0000	돈을 기다린다
0010	010 0000	돈이 위조품이 아니면 뛰여 넘는다
0011	110 1000	1000위치로 뛰여 넘기한다
0100	011 0000	총합에 돈량을 더한다
0101	100 0000	충분한 돈을 받지 못하면 뛰여 넘는다
0110	110 1010	1010위치로 뛰여넘기한다
0111	110 0001	0001위치로 뛰여넘기한다
1000	101 0000	사진을 주고 경찰을 부른다
1001	110 0000	0000위치로 뛰여넘기한다
1010	111 0000	사이다를 내주고 변화를 돌려 준다
1011	110 0000	0000위치로 뛰여넘기한다

명령은 0000위치로부터 시작해서 기억기에 련속적으로 기억된다. 먼저 하여야 할것은 오른쪽에 있는 해설대로 프로그람을 시험하는것이다. 프로그람이 무엇을 하는지 결정하기는 매우 어렵다. 매 명령들에 대해서는 PM100명령을 포함하는 표를 참고해야 하며 2진수명령을 복호화해야 한다. PM100의 원래명령을 사용해야 하므로 정확한 프로그람의 론리를 얻는것은 어렵다. 그것은 모든 항목과 련관되여야 한다.

프로그람이 무엇을 하는가는 어떻게 말할수 있는가? 프로그람작성자들은 흔히 프로그람을 《수동실행》으로 리해하거나 쓰려고 할 때가 있다. 이 수속에서 프로그람작성자는 콤퓨터처럼 움직이며 실지장치가 동작할 때 프로그람이 어떤 동작을 하는가를 확인하기 위하여 프로그람에서 명령을 읽어 내여 실행해 본다. 이 처리를 프로그람의 실행추적이라고 한다.

이것이 어떤 기술적인 작업인지 간단한 기계어프로그람의 실행을 추적하여 보기로 하자. 령에서 명령실행이 시작될것이며 사이다가격은 55쩬트(cent)이다. 프로그람의 실행추적을 표 1-5에 주었다.

1단계에서는 기계의 재설정을 진행한다. 이 작업은 받은 돈의 총합을 0으로 설정한다. 다음 0001위치에 있는 명령을 실행한다. 이 명령은 보관한 돈을 기다린다. 실례로 4분의 1이 보관되였다고 가정한다. 돈이 들어 오면 실행은 0010위치에서 계속된다.

3단계에서는 돈이 위조품인가, 아닌가를 검사한다. 돈이 위조품이 아니면 0011위치에서 명령은 뛰여 넘고 실행은 0101위치에서 계속된다. 이 명령은 총합에 돈의 값을 더하고 돈의 값이 0.25라는것을 기억한다. 그다음 프로그람은 사이다를 살만한 돈인가를 본다. 총합이 55쩬트보다 작으면 다음 명령으로 뛰여 넘으며 0111위치에서 명령이 실행된다. 이 명령은 위치 0001에로 프로그람을 이행한 다음 돈이 들어 올 때를 기다린다.

프로그람은 계속하여 다른 4분의 1과 10쩬트를 받는다(7단계부터 13까지). 그다음 돈을 받아 총합에 더한후 실행은 14단계에서 계속된다. 돈이 충분할 때 뛰여넘기는 다음명령을 수행한다. 이 명령은 위치 1010으로 이행한다. 17단계에서 사이다는 나누어 지며 나머지로서 5쩬트를 준다.

18단계에서는 1011위치의 명령이 실행되므로 프로그람은 0000위치로 돌아 가고 기계는 다른 사람을 기다리며 재설정된다.

표 1-5. 사이다기계프로그람의 수동실행

단계	프로그람계수기	동작	총 가격
1	0000	기계를 재설정한다	0.00
2	0001	돈을 기다린다. 4분의 1을 받는다	0.00
3	0010	돈이 위조품이 아니면 넘긴다	0.00
4	0100	총합에 돈량을 더한다	0.25
5	0101	충분한 돈을 받지 못하면 넘는다	0.25
6	0111	0001위치에로 뛰여 넘는다	0.25
7	0001	돈을 기다린다. 4분의 1을 받는다	0.25
8	0010	돈이 위조품이 아니면 넘긴다	0.25
9	0100	총합에 돈량을 더한다.	0.25
10	0101	충분한 돈을 받지 못하면 넘는다	0.25
11	0111	0001위치에로 뛰여 넘는다	0.25
12	0001	돈을 기다린다. 4분의 1을 받는다	
13	0010	돈이 위조품이 아니면 넘긴다	
14	0100	총합에 돈량을 더한다.	
15	0101	충분한 돈을 받지 못하면 넘는다	
16	0110	1010위치에로 뛰여 넘는다	
17	1010	사이다를 나누고 변화를 돌려 준다	
18	1011	0000위치에로 뛰여넘기한다	
19	0000	기계를 재설정한다	

기계어에서는 중간크기의 프로그람(500～1000명령)을 작성하고 검사하는데 오랜 시간이 걸리며 오유가 생긴다. 2진기계어보다 조금 높은 언어는 아쎔블리어이다. 여기서는 2진수가 아니라 기호언어지령들로 프로그람을 작성한다. 그다음 아쎔블러가 아쎔블리어명령을 2진형식으로 번역한다. 기계어와 아쎔블리어의 특징의 하나는 프로그람을 작성하는 기계의 세부를 알아야 한다는것이다. 앞에서 언급한것처럼 매 형태의 기계는 자기의 독특한 기계어를 가지고 있다.

콤퓨터언어의 다음준위는 고수준프로그람작성언어이다. 이 언어의 특징은 프로그람화된 장치의 세부적인 지식을 요구하지 않는것이다. 다른 또 하나의 특징은 이 언어가 흔히 해결하려는 문제형에 맞는 용어와 구조를 사용하는것이다. 실례로 프로그람작성언어 FORTRAN은 과학적이며 공학적인 프로그람을 푸는데 리용되는데 대수적인 표시법을 사용한다. FORTRAN은 성구 formula translation으로부터 유래되였다. 고수준프로그람작성언어에는 수백가지가 있다.

1.2 쏘프트웨어

오늘날 콤퓨터는 많은 부분에서 리용되고 있다. 그 리유의 하나는 콤퓨터가격이 훨씬 인하된데 있으며 초학자들에게까지도 쓸모 있고 리용하기 쉬운 고급쏘프트웨어들이 개발되였기때문이다.

쏘프트웨어는 응용쏘프트웨어와 체계쏘프트웨어로 구분할수 있다. 이 2개를 서로 명백히 구분하기는

힘들지만 일반적으로 응용쏘프트웨어는 특수한 문제령역이나 응용분야에서 문제해결이나 봉사를 하는데 리용된다. 응용프로그람의 범위는 대단히 크며 아주 빠르게 발전하고 있다. 참으로 개인용콤퓨터에서의 혁명은 여러가지 새로운 응용분야 즉 확장문서, 탁상출판체계(DTP), 개인정보관리체계(PIMS), 개인재정관리체계(PFMS)와 직관물관리체계(PMS)의 발전을 가져 왔다. 체계쏘프트웨어는 어떤 측면에서는 다른 프로그람의 개발과 실행을 지원하며 다른 측면에서는 응용쏘프트웨어와 하드웨어사이의 련계를 맺어 준다. 콤퓨터체계의 구성도를 그림 1-7에 보여 준다.

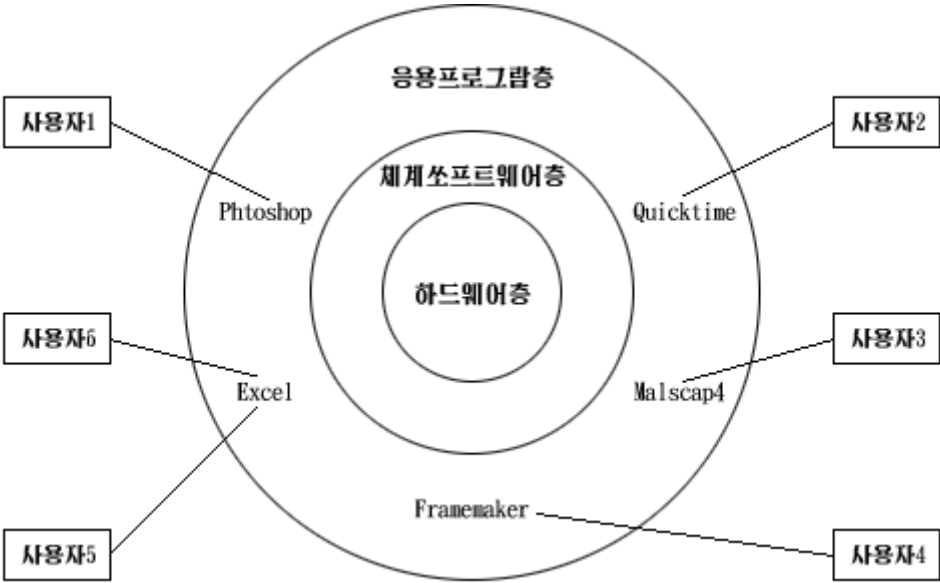



그림 1-7. 콤퓨터체계구성도

1.2.1 체계쏘프트웨어

체계쏘프트웨어에서 매우 중요한 부분은 조작체계이다. 조작체계는 장치자원을 조작하고 관리하는 쏘프트웨어이다. 이 원천은 기억기, 입출력장치, CPU를 포함하고 있다. 조작체계는 프로그람에 기억기 할당과 현시장치, 건반, 디스크구동기와 같은 입출력장치를 조종할수 있어야 한다.

개인용콤퓨터에서 대표적인 조작체계는 Windows NT, Windows 2000, UNIX 등이다. 조작체계에서 제공하는 가장 중요한 기능은 파일체계이다. 파일체계는 정보를 빨리 찾고 받을수 있게 한다. 디스크에는 서로 련관된 정보를 함께 기억하는 구역이 있다. 이 구역은 파일체계에서 목록과 류사하다. 파일체계에서 이러한 구역을 등록부라고 한다. 파일구역과 달리 등록부는 다른 등록부들을 포함할수 있다. 이러한 구성은 계층파일체계라고 부른다.

콤퓨터전문가들은 거꾸로 된 나무구조와 같은 방법으로 된 파일체계를 자주 리용한다. 디스크에 있는 파일의 계층구조를 그림 1-8에 주었다. C:은 나무의 뿌리이다. 뿌리아래에는 파일과 등록부들이 있다. 실례로 cs101은 cs101이라는 내용을 가진 파일과 등록부를 포함하는 등록부이다. 그림 1-8에서 cs101은 2개의 보조등록부 hwk와 labs 그리고 한개 파일 readme.txt를 가지고 있다. 파일이름은 등록부의 내용을 가리킨다.

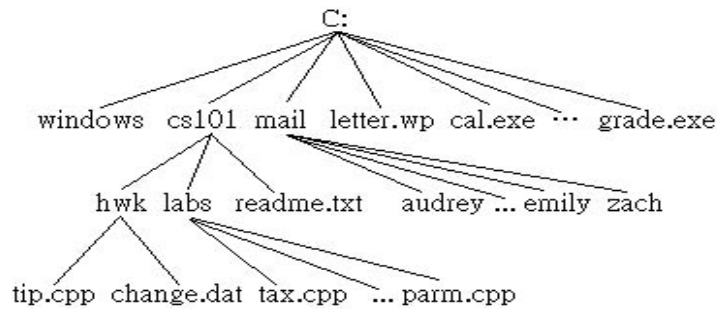



그림 1-8. 계층파일체계

이 실례에서 이름 hwk는 포함하고 있는 파일들이 cs101을 위한 과제로 되여야 한다는것을 가리킨다. 이와 류사하게 이름 labs는 학과실험실에 대한 정보를 등록부가 포함한다는것을 표시한다. 이와 같이 파일이름은 파일에 포함된 정보의 종류를 나타낸다. 파일이름은 2개의 부분을 가진다. 앞부분은 기본이름이며 뒤부분은 확장자이다. 확장자는 파일의 종류를 나타낸다. 실례로 cs101등록부에서 readme.txt는 일반확장자가 txt라는 파일을 의미한다. 이 확장자는 해당파일을 인쇄하거나 편집기를 사용하여 읽을수 있는 본문을 포함하는 파일이라는것을 가리킨다. 또 하나의 확장자는 exe이다. 이 확장자는 실행가능한 프로그람을 포함하는 파일에서 리용한다. 이러한 종류의 파일은 Word나 Notepad와 같은 본문편집기로 처리할수 없다.

보통 파일이름에는 파일에 대한 정보가 들어 있다. 실례로 readme에는 cs101등록부에 있는 파일들에 대한 정보가 들어 있다. 다른 실례로 파일 grades.xls를 고찰하자. xls확장자는 표처리프로그람인 Excel에 의해 만들어 진 파일이라는것을 의미한다. grades는 이 파일이 학년성적표를 가지고 있다는것을 가리킨다. 보조등록부에서 주의해서 파일을 만들고 알맞는 이름을 선택하여 빨리 찾기 위해 계층적파일체계를 리용하면 정보를 만드는 가장 효과적인 방법을 제공한다. 조작체계의 또 다른 중요한 부분은 파일관리명령이다. 파일에 이름을 달고 그것들을 호출하는 명령은 조작체계들마다 다르지만 그 기능은 같다. 대부분의 체계는 파일지우기, 파일이름바꾸기 ,파일복사, 등록부만들기와 같은 명령들을 가지고 있다. 조작체계는 또한 여러가지 장치에서 입출력을 수행하기 위한 기초적인 봉사를 제공한다.

이렇게 프로그람은 조작체계에 의하여 조작되므로 입출력장치의 사용법을 알아 둘 필요는 없다. 실례를 들어 프로그람이 디스크에서 어떤 파일을 읽으려고 한다면 직접 디스크에 접근하지 않고 조작체계에 요구를 보낸다.

조작체계는 디스크상에서 파일을 찾고 적당한 부분을 읽고 그다음 요구한 프로그람에 필요한 정보를 보낸다. 또 다른 하나의 조작체계봉사는 프로그람실행에 대한 관리이다. 대부분의 현대적조작체계는 다중프로그람을 CPU에 할당한다. 실례로 Windows NT조작체계는 콤퓨터의 게시판으로부터 파일을 넣을수 있으며 프로그람을 실행하는 모든 본문처리를 단번에 할수 있다.

조작체계의 과제는 매 프로그람이 실행될 때 충분한 기억기를 가지며 필요할 때 CPU를 실행하도록 하는것이다. 또 다른 체계쏘프트웨어의 한 부분은 언어처리체계이다. 언어처리체계는 쏘프트웨어를 개발하는데 리용되는 프로그람묶음이다. 번역체계의 기본내용은 언어처리프로그람작성언어로 쓴 프로그람을 읽고 CPU가 리해할수 있는 형태의 프로그람으로 출력하는것이다. 언어처리기의 입력은 원천프로그람이고 출력은 목적프로그람이다. 원천프로그람작성에 리용된 언어는 원천언어이며 또한 목적프로그람을 위하여 리용된 언어는 목적언어라고 한다. 그림 1-9에 번역과정을 보여 주었다.

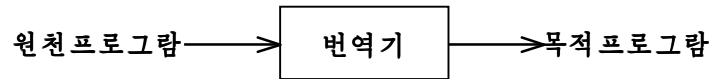



그림 1-9. 언어처리과정

언어처리기는 대체로 목적언어와 원천언어를 대상한다. 앞에서 본바와 같이 아쎔블러는 아쎔블리어프로그람을 2진수의 기계어프로그람으로 번역한다. 2진기계어프로그람을 때때로 목적코드, 또는 목적파일이라고 한다.

콤파일러도 언어처리기의 한 형태이다. 콤파일러는 고급프로그람작성언어로 씌여 진 프로그람을 번역하며 목적파일을 만든다. 고수준언어프로그람을 콤파일러를 리용하여 번역하는것을 콤파일한다고 한다. 언어처리기의 또 다른 형태는 련결기이다. 이것은 장치를 실행하기 위하여 목적파일과 서고파일을 결합한다. 서고는 여러가지 특수한 기능이나 과제들을 수행하기 위해 개발된 루틴을 위한 목적코드파일을 포함한다. 서고는 흔히 특수한 목적을 위해 제공되는데 전문적인 콤파일러의 개발자나 회사에 의해 만들어진다. 실례를 들어 입출력조작을 지원하는 서고를 들수 있다. 다른 대표적인 서고는 도형사용자대면부(GUI)를 리용하는 프로그람개발을 지원하는 루틴을 들수 있다.

이러한 서고는 창문의 열기와 현시차림표창조, 마우스와의 입출력조종을 위하여 루틴을 포함한다. 련결기의 출력은 콤퓨터로 집행할수 있는 파일이다. 적재기라는 조작체계도구에 의하여 실행가능파일은 콤퓨터의 기억기에 들어 가 실행된다.

쏘프트웨어를 개발할 때 프로그람작성자는 프로그람의 편집, 콤파일, 이미 콤파일한 목적파일과 서고파일의 련결, 넣기와 실행과 같은 동작을 주기적으로 실행하도록 예견한다.

혹시 프로그람이 정확히 동작하지 않거나 잘못 동작하면 프로그람작성자는 프로그람을 수정하여야 하며 프로그람개발을 계속해야 한다. 그림 1-10에 편집과 콤파일, 실행주기를 보여 주었다.

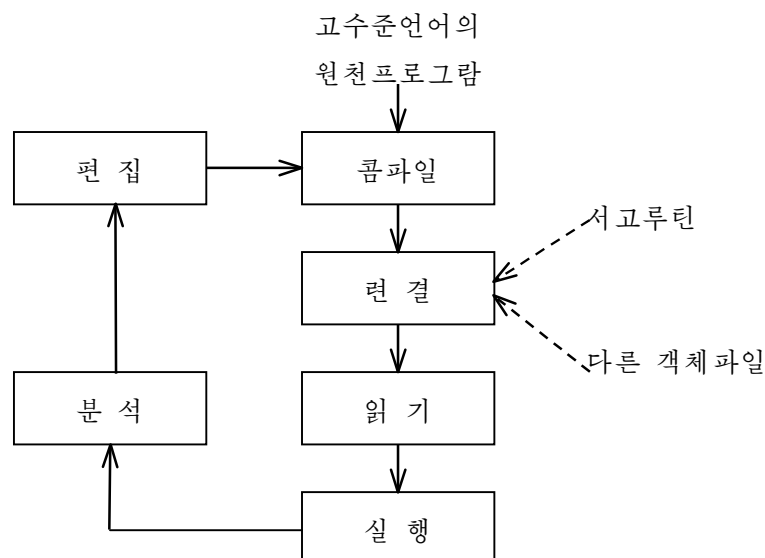
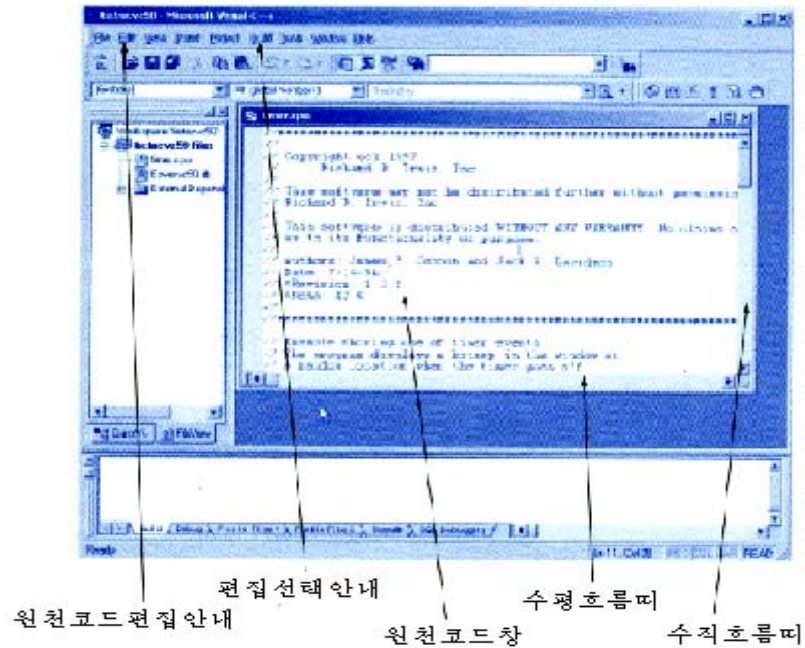



그림1-10. 편집/콤파일, 실행쏘프트웨어개발주기

쏘프트웨어가 개발되는 동안 이 주기는 계속 반복되며 따라서 언어처리기는 이 과정에 목적대로 완성되게 된다. 이 체계들을 때때로 통합개발환경(IDE)이라고 한다. 이러한 개념은 편집기, 콤파일러, 련

결기와 적재기를 통합한것이며 조종체들의 묶음은 이 과정에 촉진된다. 여러가지 IDE는 C++를 리용하여 쏘프트웨어를 개발할수 있게 한다. 고찰방법에 따라 좀 다를수 있지만 본질적으로는 같은 형태의 개발방법을 제공한다. 여러가지 차림표선택은 프로그람작성자가 지적하여 할수 있고 실행가능파일을 만들기 위하여 다른 목적파일과 서고로 그것을 콤파일하고 결과 목적파일을 련결하며 결과코드를 실행한다. 그림 1-11에 Microsoft Visual C++통합개발환경화면을 보여 주었다. IDE는 쏘프트웨어개발에서 많은 어려운 일들을 자동화하였기때문에 시간을 절약할수 있게 한다.

그림 1-11. C++ IDE

1.2.2 응용쏘프트웨어

실지로는 체계쏘프트웨어에 의하여 가공되지만 대부분의 사람들에게 필요한 도구를 만드는것은 응용프로그람이다. 응용쏘프트웨어는 형태에 따라 여러가지로 분류할수 있다.

실레로 문서편집기 Word는 문서를 만들기 위해 개발된 프로그람이다.여러가지 사무처리프로그람들인 Word Perfect, Microsoft Word 그리고 Amipro가 있다. 이 프로그람들은 편집이 간단하고 고해상도의 인쇄기로 문서를 인쇄할수 있다.

여러해동안 문서편집프로그람들이 급속히 개발되였다. 처음의 문서편집프로그람은 맞춤법검사를 포함하고 있었으나 그밖의것은 없었다. 오늘날 문법검사기, 사전, 그림도구, 표처리도구, 망접속능력 등 문서만들기를 위하여 수백가지 기능들이 개발되였다. 또한 그것들에 탁상출판프로그람(DTP)들이 개발되였다. 이 프로그람들은 문서의 편성을 지원한다. 실레로 월보나 기업소보고서를 작성하기 위한 편성도구들이 있다. 이 응용프로그람들은 다른 원천으로부터 중요한 본문과 그림을 리용하기도 한다. 문서처리기와 비교하면 DTP프로그람은 그림, 책과 같은 큰 문서를 조종하기 위한 기능들을 더 지원하고 있다.

계산의 력사

계산의 시작

계산력사의 론의에서 하나는 계산에 어떤 장치를 리용하였는가 하는것이다. 초기장치의 대부분은 셈세기와 간단한 연산을 하기 위해서였다. 셈세는데서 최초의 도구의 하나는 아직도 아이들이 쓰는 손가락이다. 초기에 인간은 손가락을 리용하여 진행하던 셈세기와 더하기, 덜기, 곱하기와 같은 단순한 연산을 수행하는 여러가지 기구를 개발하였다. 이 기구들은 완전히 복잡하였으며 큰 수를 만들게 하였다.

일부 체계들은 아시아지역들에서 리용되였다. 셈과 계산을 목적으로 고대중국은 모래에 홈을 가로 긋는 체계를 사용하였다. 값을 나타내기 위하여 자갈을 홈에 넣었다. 첫째 홈에서 한개 자갈은 하나라는 값으로 나타내며 2개 자갈은 2개라는 값을 나타낸다. 매개 련속적인 홈은 10의 제곱을 나타낸다. 따라서 첫번째 홈은 10의 0제곱,두번째홈은 10의 1제곱을 나타낸다. 따라서 두번째 홈에 자갈 2개와 첫번째홈의 자갈 3개는 값 23을 나타낸다. 후에 홈과 자갈계산기구는 우리가 알고 있는 주산과 같은 형태로 발전되였다.

그림 1-12. 주산

주산은 병렬선에 꿰여 놓은 구슬들로 구성되였다. 더하기, 덜기, 곱하기, 나누기는 적당한 구슬을 움직여 수행한다. 1940년대 후반기 주산과 콤퓨터리용자들사이의 경쟁이 분분했다. 경쟁은 산수계산을 하는것으로부터 시작하였다. 계산을 먼저 하는것이 이긴것으로 되였다. 흥미 있는것은 주산을 사용하는 사람들이 대체로 이긴것이다. 주산은 아직도 아시아와 중근동지역에서 사용되고 있다.

1.3 쏘프트웨어공학

콤퓨터의 속도가 빨라 지고 값이 눅으며 그 능력이 커짐으로써 과학자들과 학자들에게는 필수적인 도구로 리용되고 있다. 더 중요하게 생각한다면 그들은 콤퓨터를 하루생활의 한 부분으로 여긴다. 콤퓨터는 텔레비죤, 록화기, 통신수단과 함께 일반생활에 널리 쓰인다. 앞으로 은행에서 전화를 하거나 자동호출을 할 때에는 콤퓨터망에 접속한다. 콤퓨터체계가 2개의 내용 즉 하드웨어와 쏘프트웨어로 되여 있다는것을 미리 상기시킨다.

쏘프트웨어공학은 쏘프트웨어체계를 작성하기 위한 공학부문이다. 쏘프트웨어공학자의 목표는 다음과 같은 특성을 가진 쏘프트웨어체계를 만드는것이다.

- 믿음성
- 리해가능성
- 가격영향의 감소
- 적응성
- 재리용성

쏘프트웨어체계는 정확히 동작하며 오유가 없어야 한다. 문서편집기로 수행한 처리를 종이에 쓰는데는 여러시간이 걸린다. 작업도중에 비정상사태가 발생한다면 문서편집작업은 중지되며 모든 편집내용은 없어 지게 된다. 문서편집기의 오유는 피로운것이지만 보안체계가 오유를 나타냈을 때의 손실에 비해서는 가벼운것이다. 그러므로 믿음성이 있는 쏘프트웨어체계를 만드는것이 중요하다. 쏘프트웨어를 만드는

팀의 많은 성원들이 체계의 조작과 구성요소를 잘 리해하고 믿음성 있는 쏘프트웨어를 만든다면 2~3개 정도의 오유가 있을수 있다.

리해가능성도 쏘프트웨어가 오래동안 리용되는것과 관련하여 역시 중요하다. 하나의 쏘프트웨어개발은 흔히 오랜 시간이 걸리며 일단 개발된 쏘프트웨어들은 계속 오유없이 동작하여야 한다. 이러한 처리를 흔히 쏘프트웨어의 유지라고 한다. 대충 설계한 체계에 대한 수정이나 교정은 몹시 어렵다. 체계가 파괴되기전에 수정하는것이 더 효과적이다.

쏘프트웨어구입의 비용을 예측한데 의하면 그 67%가 유지에 돌려 진다. 이 비용은 쏘프트웨어체계의 설계와 조작이 알기 쉽게 되여 있을 때에 감소한다. 이러한 쏘프트웨어설계가 효과적이다. 가격과 관계되는 구성요소의 하나는 쏘프트웨어의 설치시간이다.

고객이 갑자기 쏘프트웨어제품에 대하여 요구하는 형태와 능력을 말하기는 매우 어렵다. 그러므로 합리적인 구입은 생산의 효과성이나 경쟁성을 높일수 있도록 쏘프트웨어에 대한 변화와 추가 즉 적응성을 포함한것이여야 한다. 추가의 형태와 능력보충을 간단히 할수 있도록 설계하는것은 구입비용을 감소시킬수 있게 한다.

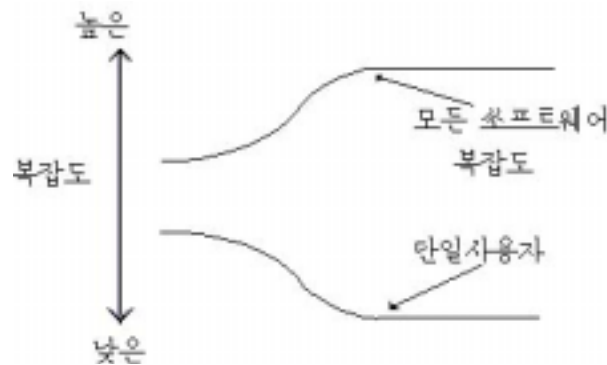



그림 1-13. 쏘프트웨어의 사용간단화에 의한 내부복잡성증가

재리용성도 프로그람가격에 영향을 주는 주요한 인자이다. 개발가격이 높은것과 관련하여 쏘프트웨어는 재리용할수 있게 만들어야 한다. 이 전략은 다른 사업에서도 흔히 리용된다. 새로운 승용차형의 설계와 제작을 생각해 보자. 자동차기사는 새차를 모두 새롭게 설계하지 못한다. 오히려 기사는 이전 승용차의 설계로부터 모방한다. 기관설계가 이전 형태로부터 일부 측면을 반복하여 리용하였다면 개발가격은 새 기관을 설계하고 시험하는것보다 감소한다. 결국 소비자의 구입가격은 낮아 지게 된다.

1.3.1 쏘프트웨어공학의 원리

쏘프트웨어공학은 참으로 복잡한 여러가지 측면들을 조절하면서 높은 성능을 가진 쏘프트웨어를 개발하기 위한 설계의 원리와 방법을 끊임없이 개척해 나가고 있다. 추상화(abstraction)란 본질적인 세부는 무시하면서 객체의 관련속성을 추출해 내는 과정이다. 추출한 속성은 객체를 정의한다.

승용차상인은 승용차를 판매형태의 표본으로 보게 한다. 관련속성은 가격, 색, 선택환경과 담보기간을 포함한다. 다른 측면에서 기술자는 구입하려는 체계의 기준으로부터 승용차를 본다. 여기서 관련되는 속성은 기름의 종류, 기름려과기의 크기, 소화전의 수와 그 형을 포함한다. 관련속성은 객체의 리용 혹은 조작방법을 나타낸다. 승용차상인에게 관련되는 승용차의 속성은 수리공에게 관련되는 속성과 차이난다(그림 1-14).

그림 1-14. 자동차에 대한 두사람의 보기 혹은 추상화

관련속성에 대한 고찰과 적합치 않은 세부의 무시에 의해 객체와의 교섭에서 복잡성은 감소한다. 추상화는 쏘프트웨어의 설계와 쓰기의 복잡성을 관리하는데서 기본이다. 실례에서와 같이 디스크상에서 파일을 찾고 읽어 내는것을 생각해 보자. 만일 특수파일을 찾는 방법과 읽는 방법을 조종해야 한다면 이 과제수행은 매우 어렵다.

우리는 구동기를 조작하는 저수준명령으로 자료를 디스크상에 보관하는 방법을 알고 있다. 파일체계는 이러한 저수준체계를 무시하기 위해 디스크상에서 정보의 추상적호출을 제공한다.

파일을 간단히 파일의 이름을 주어 호출할수 있다. 파일체계는 디스크구동기에서 자료읽기의 저수준세부를 조종하며 귀환은 프로그람으로 한다.

정보은페 즉 교갑화(encapsulation)는 객체를 외부와 내부로 가르는 파정이다. 객체의 외부는 체계에서 다른 객체를 보거나 알수 있게 한다. 외부는 체계의 다른 부분에 영향을 주지 않는 세부이다. 객체의 내부교갑화는 체계의 다른 부분에 영향이 없이 객체을 변화시킨다는것을 의미한다.

자동차를 실례로 본다면 승용차에서 라지오를 생각해 볼수 있다. 라지오의 외부는 전자장치, 고성기, 안테나와 라지오를 접속할수 있는 여러가지 조종기와 접속기들이다. 라지오의 내부는 라지오가 어떻게 동작하는가 하는 세부이다(그림 1-15).

승용차에 라지오를 설치하고 사용하기 위하여 전자공학에 대해서 알아야 할 필요는 없다. 본질적으로 라지오는 단추와 케블이 있는 검은 통으로 볼수 있다. 정보은페의 큰 우점은 복잡한 체계를 변화시킬수 있게 해주는것이다.

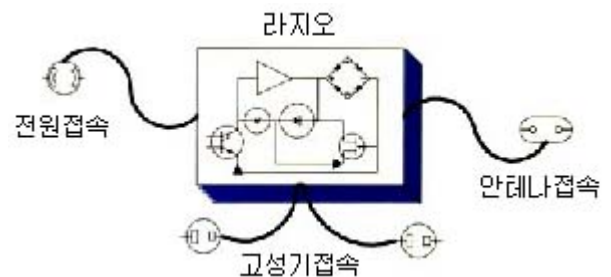



그림 1-15. 승용차라지오의 교갑화

승용차의 라지오는 승용차의 다른 구성요소들에는 영향이 없이 CD구동기를 포함하여 바꿀수 있다. 라지오의 조작이 교갑화되여 있고 라지오의 외적보기와 조절기들이 접속구들에 의하여 규정되므로 낡은 라지오를 쉽게 뽑아 내고 새것을 쉽게 련결할수 있다.

체계의 설계에 비유한다면 교갑화는 체계의 다른 부분에는 영향이 없이 변화시키는 쏘프트구성요소의 내부조작을 허락하게 한다. 실례로 교갑화의 원리가 자동음성우편체계에 정확히 적용된다면 체계의 다른 부분에 영향이 없이 통보문보관을 조종하는 구성요소를 변화시킬수 있다.

실례로 우리는 사용자가 보관할수 있는 통보문의 수가 증가할것을 요구한다. 만일 통보문보관체계를 숨기고 분리시킨다면 이 변환은 사용자가 체계를 호출하고 통보문을 받는데 영향을 주지 않는다. 모듈화는 목표를 실현하기 더 쉽게 하기 위해 더 작은 부분 혹은 모듈안의 객체를 나누는 처리이다. 실례로 매

구성요소들을 개별적으로 시험할수 있게 하기 위해 구성요소안에 복잡한 객체을 만든다. 자동차가 조립되면 기관, 동력, 전달장치와 라지오와 같은 여러가지 구성요소들이 개별적으로 시험된다. 모듈화는 완성된 승용차의 시험시간과 승용차가 잘못 조립될 가능성을 줄인다. 이와 같이 구성요소를 쉽게 재리용하기 위해서 하나의 객체를 만든다. 대부분의 복잡한 체계는 모듈식으로 되여 있다. 그것들은 결합된 더 간단한 작업요소나 묶음으로 조립된다. 복잡한 체계의 정확한 모듈화는 복잡성을 피하는데 도움을 준다.

물체를 리해하기 위하여 더 작고 쉽게 풀어 나가는것은 다른 체계를 더 쉽게 리해하게 한다. 실례로 자동차는 보조체계들로 분해할수 있다(그림 1-16).

그림 1-16. 자동차의 보조체계

자동차보조체계에는 랭각체계(방열기, 물뽐프, 자동온도조절장치 등) 점화체계(전지, 기동장치, 점화전 등) 그리고 배기체계(반응변환기, 소음장치 등)가 포함된다. 자동차들을 이처럼 분해하여 생각한다면 전체 구조와 조작을 쉽게 알수 있다.

여러가지 관계에 기초한 객체의 등급이나 순서는 계층형이다. 등급체계는 복잡한 조직과 체계를 리해할수 있게 한다. 그림 1-17에 대표적인 회사의 기구도표를 주었다. 도표는 누가 누구에게 복종하는가 하는 관계에 기초한 종합원등급체계를 보여 준다. 회사등급체계는 종업원들이 자기의 위치를 알수 있게 한다.

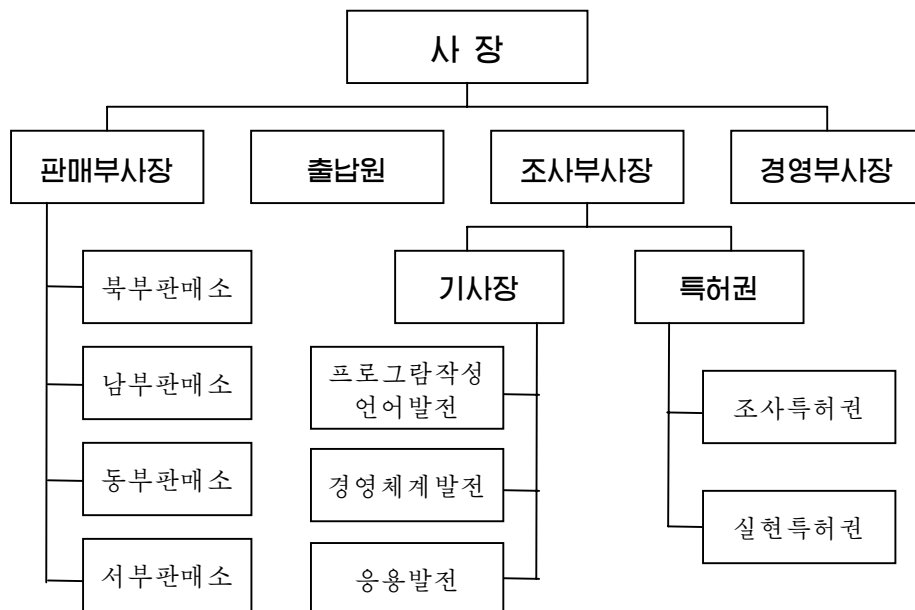



그림 1-17. 회사의 기구도표

추상적으로 구성된 복잡한 체계를 위해 그와 류사한 체계를 배렬하는데서 가장 좋은 방법은 가장 간단한것으로부터 일반적인것까지 가는것이다.

과학자들은 식물과 동물분야의 종을 판단하고 분류하는 기술을 오래전부터에 리용하여 왔다. 자연관계에 기초한 등급체계순위를 분류법이라고 한다. 이러한 등급체계는 일반적으로 특성과 행동의 관계를 로출시키기때문에 모든 추상을 더 쉽게 리해할수 있다.

그림 1-18에 공룡의 분류법을 주었다. 공룡은 골격구조에 따라서 2개 부분으로 나눌수 있다.

Saurisching (도마뱀골격공룡) 부류는 Tyramosamurus와 Velociraptor와 같은 화석공룡을 포함하며 Ornithischia(익룡)부류는 Stegosaur와 Trceratops와 같은것들을 포함한다.

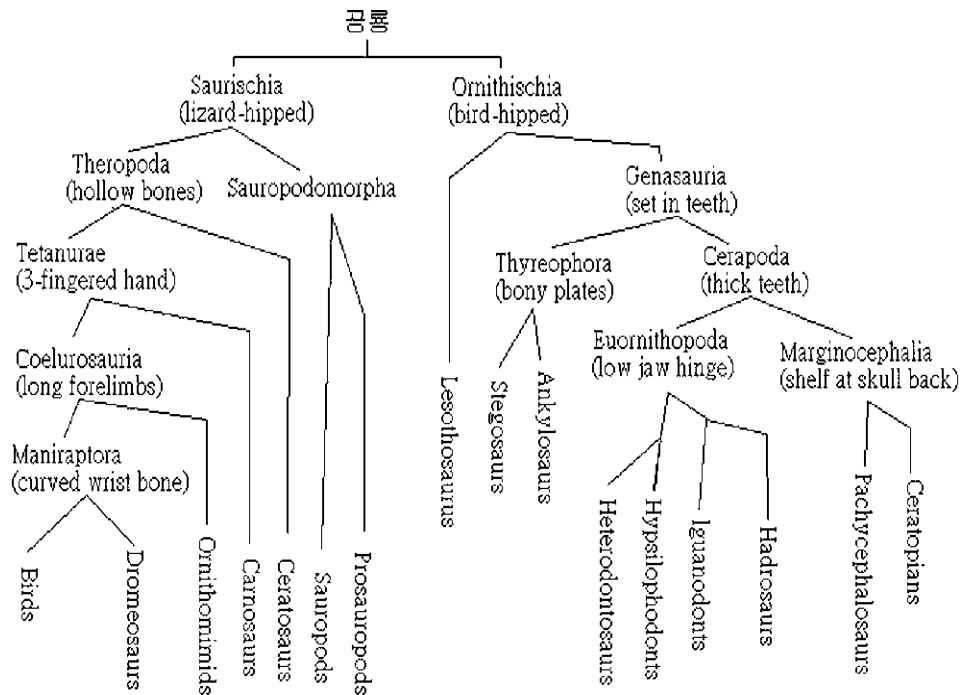



그림 1-18. 공룡분류법

1.4 객체지향설계

앞에서 서술된 원리들을 반영한 프로그람설계와 작성방법문제들이 여러가지로 발전하여 왔다. 최근에 시작된 객체지향설계 및 프로그람작성법은 쏘프트웨어개발자들로 하여금 믿음성이 높고 가격이 낮으며 적응성, 리해가능성, 재리용성을 실현하기 위한 목표를 달성할수 있게 한다. 객체지향쏘프트웨어설계는 우리가 생각하는 방법을 명백히 현대화한 방향에서 쏘프트웨어를 고찰하는 오늘의 세계적추세에 맞는다.

이제 객체에 대하여 간단히 언급하고 넘어 가자. 실례로 애기의 울음은 모든 소음에 대하는 속성이라는것을 알수 있다. 천성적기질을 계발시킨후에 객체가 속성을 가진다는것을 알고 추상적으로 생각하기 시작한다. 실례로 애기의 울음은 모든 소음에 대한 속성이라는것을 알수 있다.

속성을 가진 객체로서의 세계를 보는 실례는 우리가 여러가지 복잡한 문제들을 고찰하는데서 도움을 주며 해야 할 많은 일들을 생각해 볼수 있게 한다. 집에 앉아서 텔레비죤을 통하여 여러 통로를 쉽게 볼수 있다(그림 1-19). 이때 알맞는 통로를 선택하기 위하여 원격조종, 단추누르기 등을 리용한다. 이 동작을 분석해 보면 먼저 원격조종기를 찾아 볼수 있는데 이것은 물리적객체로 볼수 있다. 이 객체는 무게와 크기와 같은 속성을 가지며 또한 여러가지 기능을 수행할수 있다. 그것은 텔레비죤에 통보를 보낼수 있다. 이것을 어떻게 하며 통보를 어떻게 부호화하겠는지는 명백치 않지만 그것을 알 필요는 없다. 오직 수와 단추만을 알면 된다. 단추는 원격조종대면부이다. 객체의 대면부를 리해한다면 객체의 동작을 알지 못하고도 여러가지 과제를 수행하는데 그것을 사용할수 있다. 해당한 단추를 누르면 텔레비죤에 통보를

보내는 원격신호를 발생시킨다. 텔레비죤도 역시 물리적객체이다. 원격신호에 의한 통보를 접수함으로써 텔레비죤은 요구되는 통로로 절환된다.

그림 1-19. 객체호상작용 즉 통보

이러한 호상작용은 놀라우리만큼 일반적이다. 여기서 두 객체의 호상작용을 설정할수 있으며 객체들의 내부동작에 대해서는 리해함이 없이 복잡한 호상작용을 수행하게 할수 있다. 두 객체들에 대한 추상화가 이와 같은 결과를 가져오게 되는것이다. 텔레비죤과 원격조정기에 대한 추상화는 다른 집에 가서 다른 상표가 붙은 원격조절기와 텔레비죤을 얼마든지 사용할수 있다는것을 의미한다.

같은 객체는 같은 동작을 나타낸다. 주위세계와의 호상관계에 대한 이러한 고찰방법은 프로그람작성에도 적용할수 있다.

객체지향설계를 리용하여 복잡한 체계를 개발하는데서 기본단계는 체계를 구성하는 객체들을 결정하는것이다. 이 객체들에 대한 적당한 추상화와 외적동작과 내적동작을 분리하여 고찰하는데 주의를 돌리면 복잡하고 큰 쏘프트웨어체계를 설계하는데 도움이 된다.

객체로 무엇을 표현할수 있는가? 사실 물리적인것들은 다 객체이다. 공, 파일등록부, 주소책, 나무, 콤퓨터 등은 모든 객체이다. 수자, 단어, 계산자리나 악보책과 같은것들은 물리적인 객체들은 아니지만 속성을 가지고 동작을 수행하므로 객체라고 볼수 있다.

하나의 수는 어떤 값을 가지며 동시에 두개의 수는 더해 질수 있다. 또한 단어는 한개의 단위이며 문서편집기에 대하여 말한다면 단어를 문서에 추가하거나 삭제할수 있다. 음악에서 음은 음량, 지속성 음높이 등의 속성을 가진다.

거의 모든 부분에서 이름, 그와 관련한 속성, 그것을 리해할수 있는 통보에 의하여 객체들을 정의할수 있다.

일반적으로 객체가 하나의 통보를 받으면 그에 따라 여러가지 동작을 하거나 속성들중의 어느 하나를 변화시켜 새로운 객체를 만든다. 원격조정실례에서 텔레비죤이 《통로변환》통보를 받으면 통로가 절환된다. 만약 쏘프트웨어개발에 객체지향방법을 적용하려고 한다면 객체라는 관점에서 생각하고 수행결과를 제공하는 프로그람작성언어를 사용하여야 한다. 이러한 프로그람작성언어는 객체지향프로그람작성언어이다.

객체지향설계를 수행하기 위하여 객체지향프로그람작성언어를 사용하는것을 객체지향프로그람작성이라고 한다. 《객체지향설계를 수행한다.》라는 문구를 매우 주의깊게 리용하여야 한다. 후에 볼수 있겠지만 객체지향언어를 실제로 사용하면서도 객체라는 관점은 얼마 가지지 못하는 페단이 있다. 현재 대표적인 객체지향프로그람작성언어는 Smalltalk, C++, Eiffel 그리고 Java 등이다. 이 언어들이 제공하고 있는 객체지향프로그람의 형태는 대체로 갈다. 다만 이 언어의 객체와 문법에 대한 술어에서 차이가 있을뿐이다.

1.4.1 객체지향프로그람작성

객체지향프로그람작성언어의 여러가지 형태를 보여 주기 위하여 BUG HUNT(벌레잡이)라는 간단한 콤퓨터유희의 설계를 실례로 들어 보자.

이 유희프로그람의 목적은 다름이 아니라 사람들이 마우스로 자리표를 지정하는 능력을 키워 주는데 있다. 유희는 다음과 같이 동작한다. 움직이는 벌레가 화면의 창문에 나타난다. 벌레의 방향은 임의로 변화된다(그림 1-20). 유희의 목표는 벌레를 잡아 치우는것이다. 벌레는 마우스지시자를 벌레우에 가져다 대고 찰칵하여 없앨수 있다. 첫 벌레가 없어 지면 또 다른 벌레가 생겨 난다. 만일 벌레를 놓쳐 버리면 사용자는 유희에서 지는것으로 된다. 느린 벌레와 빠른 벌레를 둘 다 놓치지 않고 잡아 치웠을 때 이긴것으로 된다.

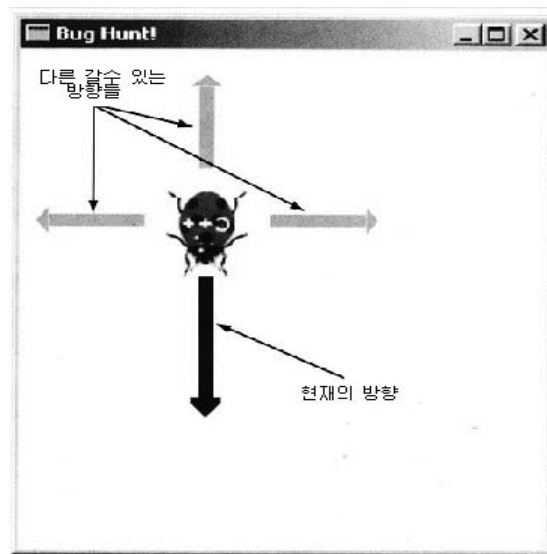



그림 1-20. Bug Hunt 유희

프로그람의 설계를 시작하기전에 프로그람을 작성하기 위한 정확한 구상을 서술해야 한다. 이때 벌레의 화상을 포함한 하나의 창문을 설정하고 벌레가 이 창문에서 임의의 방향으로 움직이며 느린 벌레와 빠른 벌레 두가지 있다고 보겠다.

느린 벌레는 빠른 벌레가 움직이는것보다 더 느리게 움직인다는것은 명백하다. 이 두 벌레는 서로 다른 방식으로 움직인다. 느린 벌레가 창문의 변두리를 돈다면 마치도 벌레가 벽에 밀리우는것처럼 방향을 바꾼다. 빠른 벌레가 창문변두리를 돈다면 창문변두리에 부딪쳐 다른쪽 변두리로 튀여 났다가 다시 튀여 나는 식으로 왔다갔다한다.

벌레를 잡기 위하여 사용자는 마우스지시자로 벌레를 지시하고 찰칵한다. 지시자가 벌레를 정확히 지시하지 못하였을 때에는 벌레는 빠져 나간다. 이때 마우스단추를 찰칵하면 튀여나오기창문에서는 사용자에게 벌레를 놓쳤다고 알려주고 유희를 다시 시작하도록 한다. 벌레를 잡아 치우기 위하여 다시 마우스로 찰칵하는 동작을 여러번 해야 한다. 빠른 벌레를 잡아 치웠을 때 유희는 끝난다.

우의 유희프로그람에서는 객체에 알맞는 기능으로서 창문, 마우스, 벌레를 정할수 있다.

객체지향언어들가운데서 리용할수 있는 형태를 보여 주기 위해 두가지 객체의 설계에 초점을 둔다. 창문과 마우스의 설계를 련습 삼아 시작해 보자. 매개 벌레와 관련한 속성들과 그것이 수행할수 있는 동작을 결정한다. 벌레의 속성은 다음과 같이 결정한다.

- 창문에서의 위치
- 화상 즉 그림에 의한 벌레의 보임새
- 현재속도
- 현재방향
- 힘(즉 벌레를 잡기 위하여 마우스를 찰칵하는 회수)

벌레는 다음의 통보와 지령에 의하여 조종되여야 한다.

- 그리기
- 벌레이동(즉 현재위치의 변화)
- 벌레의 이동에서 방향변환
- 벌레의 잡기(즉 벌레를 잡았다는것을 알려 준다)
- 벌레의 죽이기(즉 벌레를 죽게 한다)

마우스지시자가 벌레의 가운데를 정확히 지시하고 있는가를 알아 본다. 이러한 속성과 통보로부터 벌레잡이에서 벌레에 대한 추상화가 실현된다. 객체지향언어는 하나의 수법으로서 숨은 속성과 통보에 의한 추상화의 방법을 제공한다. 이러한것을 일반적으로 클라스라고 한다.

속성과 통보묶음이 클라스에 감추어 져 있으므로 그것들을 흔히 클라스의 성원이라고도 한다. 클라스의 성원속성들은 성원들에 대한 정보를 가지고 있으므로 자료성원이라고 한다. 객체의 클라스를 조정할수 있는 통보는 흔히 메쏘드 또는 성원함수라고 한다.

클라스와 객체사이의 차이를 잘 리해하는것은 아주 중요하다. 클라스는 추상화된 개념이지만 객체는 구체적개념이다. 실례로 승용차라는 개념은 클라스이지만 차체내부와 기판은 객체이다. 형상적으로 말한다면 클라스는 객체의 인쇄판 혹은 형타처럼 생각할수 있다.

클라스로부터 속성의 정의도 찾아 내고 객체를 창조하고 증시할수 있다. 클라스추상화로부터 구체적인 객체에 대한 증시의 일반리해를 그림 1-21에 주었다.

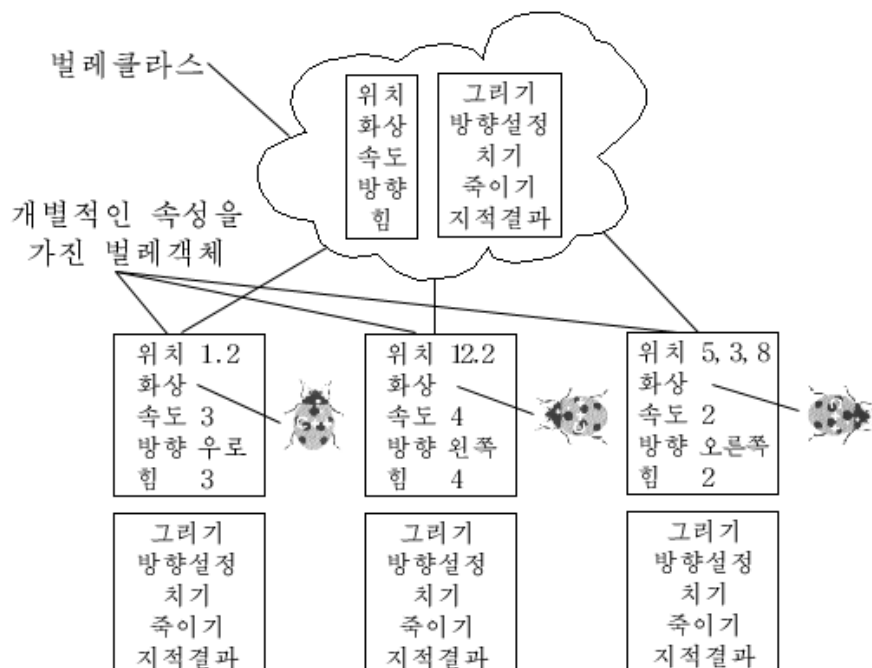



그림 1-21. 벌레클라스로부터 두가지 벌레객체의 증시

여기에 서로 다른 세마리의 벌레가 있는데 그것들은 각기 자기의 위치, 모양, 운동방향 그리고 힘을 가지고 있으며 이것이 벌레클라스에 대한 설명이다. 구체적인 객체로서 지정된 클라스를 식별한다. 본질상 클라스는 객체의 속성과 통보를 정의한다. 구체레에서는 매 속성값들을 정의하여 객체를 창조한다. 이렇게 하여 매개 벌레(느린것과 빠른것)에 따르는 클라스를 창조할수 있으며 요구하는 여러가지 종류의 벌레를 정의할수 있다.

그런데 객체지향언어에서 중요한것은 프로그람작성자가 하나의 객체에 대하여 일관성있는 정의를 하는것이다. 이를테면 빠른 벌레와 느린 벌레는 각기 위치, 속도, 모양, 움직이는 방향, 힘이 일관하게 정의되여야 한다. 사실 느린 벌레와 빠른 벌레를 식별하는것은 그것들이 어떻게 움직이는가 하는데 따른다. 느린 벌레는 창문의 변두리를 칠 때 밀리우는 식으로 방향을 바꾸지만 빠른 벌레는 빈 자리를 인차 내고 창문의 반대변두리에 다시 나타난다. 이렇게 일반적으로 모든 벌레가 가지고 있는 속성들과 동작들을 얻어서 기본벌레를 만들며 창문의 변두리를 칠 때 서로 다르게 동작하는 느린 벌레와 빠른 벌레 두가지를 만든다. 이처럼 느린 벌레와 빠른 벌레는 서로 다른 종류의 객체이다.

흔히 《is-a(대용)》관계로 되여 있는 이러한것이 하나의 체계를 이룬다. 그림 1-22에 이 체계를 보여 준다.

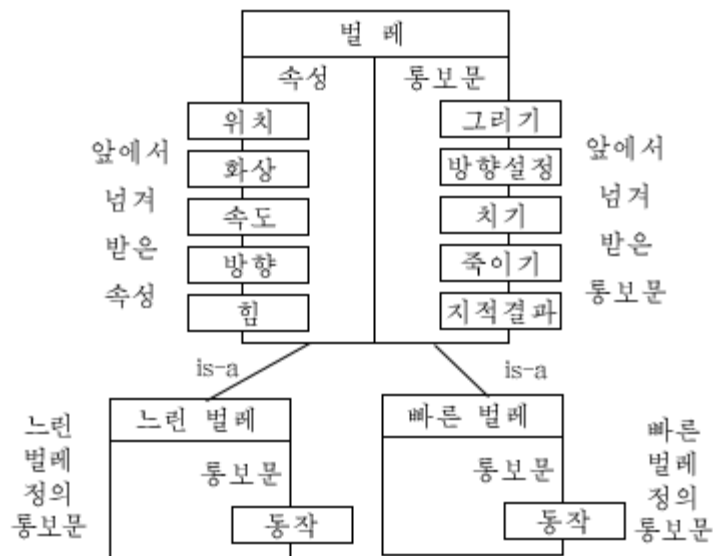



그림 1-22. 각이한 동작에 의한 벌레의 추상화와 체계

도표의 맨 꼭대기에는 벌레클라스가 있다. 가장 일반적인 이 클라스를 흔히 기초클라스 혹은 상위클라스라고 한다. 기초클라스의 아래를 보조클라스 혹은 파생클라스라고 한다. 파생클라스는 기초클라스의 속성들과 통보들을 계승한다. 이렇게 느린 벌레와 빠른 벌레는 둘 다 위치, 모양, 속도, 운동방향 및 힘과 같은 속성들을 가지고 있다. 이와 같이 빠른 벌레와 느린 벌레는 그리기, 방향설정, 치기, 죽이기와 같은 통보들을 인식하며 마우스지시자가 그것들을 지시하고 있는가를 느낀다.

계승이라는 일반개념은 객체지향언어에서 쓰이는 특징적인 개념이다. 기초클라스로부터 속성과 성원함수들을 계승하는 클라스는 재리용원리를 제공한다. 두가지 형태의 벌레는 그리기, 방향설정, 치기, 죽이기 및 힘과 같은 성원함수들을 함께 가지고 있다. 이것들은 림시로 수행될수 있다.

벌레에 대한 클라스체계는 객체지향언어의 또 다른 하나의 특징인 다형성에 대한 개념을 가지고 있다. 다형성은 여러가지 형태로 변형될수 있는 능력을 말한다. 객체지향언어에서 다형성은 통보와 그것을

받는 객체가 여러개이라는것을 반영한것이다. 통보의 변형은 느린 벌레인 경우에는 하나로 볼수 있고 빠른 벌레인 경우에는 여러개라고 볼수 있다. 느린 벌레는 창문의 변두리에 부딪칠 때 밀리는 식으로 방향을 바꾸지만 빠른 벌레는 다른쪽으로 옮겨 간다. 다형성의 이러한 실례는 빠른 벌레와 느린 벌레에 대한 통보의 변형을 비롯하여 그림 1-22에서 보여 주었다.

다형성은 객체를 리용하기 위한 보편적개념이다. 흔히 류사한 객체들은 같은 통보를 받아도 서로 다른 행동을 한다. 실례로 콤퓨터의 도형사용자대면부를 보자. 여기에는 파일을 표시하는 객체들인 여러가지 아이콘들이 있다. 이 객체들에 마우스로 통보를 보낸다. 일반적으로 통보는 두번찰칵으로 보내 지는데 이때 마우스지시자를 객체우에 가져다 놓고 두번 찰칵하여 그 객체에 사건을 보낸다.

실행가능파일에 대한 두번찰칵은 프로그람실행을 의미하며 본문파일에 대해서는 본문편집기의 기동과 편집하기 위한 파일열기를 한다는것을 의미한다. 벌레체계의 개발은 객체지향언어에서 사용할수 있는 많은 형태를 설명한다. 그런데 객체지향설계와 프로그람작성의 힘을 완전히 론증하기 위해서는 객체묶음으로부터 완성된 체계를 만들수 있는 방법에 대하여 론의하여야 한다. 벌레잡이의 고수준설계를 묘사하여 이것을 할수 있다. 다른 객체들로는 마우스와 창문이 있다. 이 객체들에 대한 실현도 필요하다.

그런데 또 다른 매우 중요한 객체가 설명에서 언급되였는데 그것이 바로 유희이다. 문제설명이 이 객체에 대한 설명이므로 검사해 보기는 쉽다. 추상적인 고찰로부터 유희객체는 다른 객체들의 활동력을 보여 주는것이며 유희의 원리가 다르다는것을 확신할수 있다. 유희는 유희조종자라는 객체를 호출한다.

이제는 벌레잡이의 고수준서술을 완성할 준비가 되였다. 언급한바와 같이 실지 마우스와 창문객체가 요구된다. 지금은 창문객체를 무시할수 있다. 창문의 창조와 조종은 유희의 수행에서 필요하지만 유희에서 기본역할을 놀지는 않는다.

다른 한편으로 마우스는 유희에서 중요한 요소이다. 유희의 기본동작은 마우스에 의해 진행된다. 마우스객체에 대한 추상적고찰은 유희조종자에 통보를 보낼수 있다는것이다. 마우스는 단추를 누를 때마다 유희조종자에 통보를 보낸다.

통보는 마우스지시자의 화면위치를 담고 있다. 마우스와 벌레에 대한 추상적측면에서 벌레잡이의 설계와 조작은 완전히 간단하다. 모든 조작을 그림 1-23에 보여 주었다. 마우스단추를 누르면 마우스는 유희조종자에 마우스눌림통보를 보낸다. 유희조종자는 이 통보로부터 마우스위치를 검출하고 화면에서 벌레에 대한 마우스위치를 포함한 지정결과통보를 보낸다. 벌레는 통보에서의 위치가 자기안에 있는가를 결정한다.

만일 마우스가 벌레를 지적하였다면 지정결과 통보문에 Yes로 응답하고 아니면 No로 응답한다. 만일 유희조종자가 Yes를 받으면 유희조종자는 벌레에게 치기통보를 보낸다. 벌레의 힘이 조금 있었다면 벌레는 치기통보에 힘이 없다고 응답한다. 이 응답을 받으면 유희조종자는 벌레를 죽인다. 만일 느린 벌레였다면 유희조종자는 빠른 벌레를 만들고 유희를 시작한다. 유희조종자와 벌레사이의 호상작용을 그림 1-23에서 보여 주었다. 만일 벌레가 응답이 없으면 마우스는 벌레를 지정하지 못한것으로 되며 유희조종자는 새로운 창문을 만들어 사용자가 벌레를 놓쳤다는것을 알린다. 모든 벌레가 다 없어 지면 유희조종자는 마우스숙련솜씨를 축하하여 창문에 축하통보를 내보낸다.

물론 벌레잡이를 수행하자면 아직도 많은 문제들이 있다. 그러나 높은 수준에서의 설계는 15장에서 계속 취급한다.

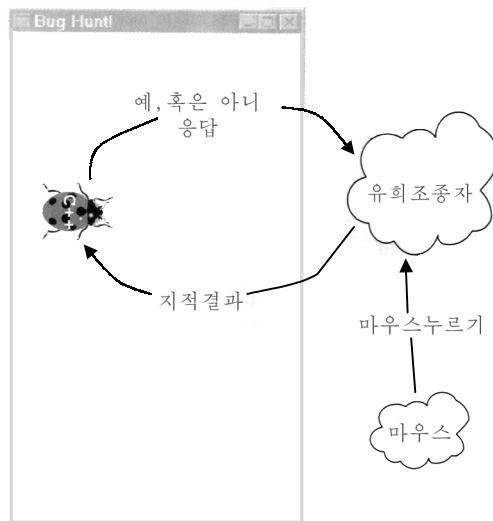



그림 1-23. 서로 다른 동작에 의한 벌레의 추상화와 체계

콤퓨터의 력사

알고리듬

정신적으로 또는 장치로 세거나 계산하는 임의의 체계는 다음과 같은 단계나 방향을 가진다. 콤퓨터학자들은 이러한것을 서술하기 위하여 알고리듬이라는 말을 리용한다. 이 말은 아주 흥미 있게 유래되였다. 9세기초에 칼리프 마맘이라는 사람이 바그다드에 큰 고등교육쎈터를 세웠다. 쎈터는 윈덤 하우스로 불리 웠다. 한 학자는 《기억의 원리와 감소》라는 매우 흥미 있는 책을 썼다. 대수학이라는 말은 이 책의 제목에서 유래되였다. 이 책에서는 힌두수자를 소개하였고 옹근수에 의한 기본연산의 체계를 설명하였다. 이전 아랍제국과 그후 제국들에 의해 힌두수자의 사용이 촉진됨으로써 그 책은 더욱 유명해 졌다. 알고리듬이라는 말은 바로 학자의 이름 Abu Jafar Mohammed ibn Musa al - Khowarizmi에서 유래되였다. 잘 알려 져 있는 알고리듬의 실례는 유클리드알고리듬인데 2개의 옹근수 m과 n의 최대공약수(GCD)를 계산하는 과정이다. 알고리듬은 다음과 같다.

걸음1: m을 n으로 나눈 나머지를 r로 한다.

걸음2: r가 0이면 알고리듬을 끝낸다. N은 GCD이다. 0이 아닌 경우 m=n으로, n=r 로 하고 걸음1로 돌아 간다.

알고리듬의 사용을 증명하기 위하여 12와 8의 GCD를 계산하자. 매 단계를 련이어 계산한다.

단계1: 걸음1 즉 r=4(12를 8로 나눈 나머지)

단계2: 걸음2 즉 r가 0이 아니므로 m=8, n=4로 하고 단계1로 간다.

단계3: 걸음1 즉 r=0(8을 4로 나눈 나머지)

단계4: 걸음2 즉 r=0이므로 알고리듬을 끝낸다.

12와 8의 GCD는 4이다.

알고리듬은 단계들의 모임으로서 표현된다. 매 단계는 주어 진 여러가지 동작을 서술한다. 이 알고리듬에서 매 단계에 수행되는 동작을 서술하는데 자연언어가 쓰이였다. 대부분의 알고리듬에서 단계들은 자연언

어와 수학개념의 결합으로서 서술된다. 단계를 서술하는데 사용되는 개념은 그리 적합치는 못하지만 중요한 것은 정확한 동작을 서술하는것이다. 이 책의 뒤장에서 특수한 과제를 수행하는 프로그람을 만들기때문에 흔히 과제를 수행하는 방법을 알고리듬으로 서술한다.

문 제

1. 일반적인 관계에 기초한 체계배렬은 무엇인가?
2. 고수준언어프로그람을 기계어코드로 번역하는 프로그람은 무엇인가?
3. 목적파일과 서고파일을 결합하여 하나의 단위로 만드는 프로그람의 이름은 무엇인가?
4. 비본질적인 세부는 무시하고 객체와 관련되는 속성을 확인하는 처리가 무엇인가?
5. 객체를 내부와 외부로 가르는 처리는 무엇인가?
6. 객체를 작은 부분, 모듈로 갈라서 여러개의 목표를 얻기 쉽게 하는 처리는 무엇인가?
7. 통보를 받는 객체가 여러개라는것을 나타내는 객체지향속성은 무엇인가?

1.5 알아 둘 점

✓ 콤퓨터의 속도는 보통 초당 주기로 평가한다. 일반적으로 장치는 초당 100～200MHz 혹은 100만～200만주기에서 동작한다.
✓ 콤퓨터는 2진수체계를 사용한다. 2진수는 한개 비트를 쓴다.
✓ 콤퓨터에서 기억의 기본단위는 byte 혹은 8bit이다.
✓ 부수는 보통 2의 보수형식으로 기억된다. 이 형식에서 nbit수는 -2^{n-1}～$2^{n-1}-1$범위의 값을 표현할수 있다.
✓ 중앙처리장치 CPU는 콤퓨터의 중심이다. 산수연산과 론리연산을 진행한다.
✓ RAM의 크기는 MB로 측정한다. 현재 탁상콤퓨터는 64～512MB범위의 기억기를 가지고 있다.
✓ 하드디스크의 용량은 대체로 10～75GB정도이다.
✓ 프로그람언어는 콤퓨터에 명령을 주는 언어이다.
✓ 콤파일러는 프로그람작성언어를 기계어로 번역한다. 기계어는 콤퓨터가 직접 실행할수 있는 연산을 포함한다.
✓ 응용쏘프트웨어는 특수한 문제를 풀거나 특수한 봉사를 하는 쏘프트웨어이다.
✓ 체계쏘프트웨어는 다른 프로그람의 개발과 실행을 제공하는 쏘프트웨어이다.
✓ 조작체계는 기억기, 입출력장치, CPU와 같은 콤퓨터자원을 조종관리하는 체계쏘프트웨어이다.
✓ 정보는 파일형식으로 디스크에 기억되므로 빨리 호출할수 있다.
✓ 알고리듬은 여러가지 과제를 수행하는 방법에 대한 세부적인 단계별 서술이다.
✓ 쏘프트웨어전문가의 목표는 믿음성 있고 리해할수 있으며 가격이 눅고 적응성이 좋을뿐아니라 재리용할수 있는 쏘프트웨어를 개발하는것이다.
✓ 추상화는 비본질적이거나 적합치 않은 세부들을 무시하고 객체들의 본질적이며 고급한 분야를 포괄하는 과정이다.
✓ 교갑화 즉 정보은폐는 객체의 외부를 가르는 처리인데 다른 객체로부터 보거나 호출될수 없으며 내부수행세부는 다른 객체들로부터 숨길수 있다.
✓ 모듈성은 객체를 작은 부분으로 가르고 더 작은 객체나 모듈이 개별적으로 관계할수 있게 하는 처리

이다.

- ✓ 계승체계는 가장 일반적인것으로부터 덜 일반적인것까지 추상을 하나로 묶는 방법이다. 객체지향설계와 프로그람작성은 다른것들과 호상작용하는 객체들의 모임으로서 쏘프트웨어체계를 형태화한 프로그람작성의 대표적실례이다.
- ✓ C++에서 추상화는 클라스창조에 의하여 진행된다. 클라스는 객체의 동작과 속성을 은폐시킨다.
- ✓ 클라스의 작업성원은 클라스의 특성이나 속성들이다. 클라스의 성원함수는 클라스의 동작이다. 기초클라스는 보다 특수한 클라스가 파생될수 있는 클라스이다. 파생클라스는 기초클라스의 속성을 계승한다.
- ✓ 다형성은 여러가지 형태를 가정할수 있는 능력이다. 객체지향언어에서 다형성은 통보를 받는 객체형에 의존하는 여러가지 성원함수에 의해 제공된다. 구체례제시는 클라스추상화로부터 구체적인 객체를 만드는 과정이다.

참고할 책

다음의 책들은 콤퓨터의 력사에 대하여 더 잘 알수 있게 한다.

Stan Augarton, Bit by Bit: An Illustrated History of Computers, New York: Ticknor & Fields, 1984

Jon Palfreman and Doron Swade, The Dream Machine:Exploring the Computer Age, London: BBC Books, 1991

Harry G.stine, The Untold Story of the Computer Revolution,NewYork: Arbor House,1985

Michael R.Williams, A History of Computing Technology, Los Alamios, CA: IEEE Computer Society Press, 1997

련습문제

1.1 극소형처리소자의 동기속도가 120MHz라고 하자. 만일 더하기연산을 한 박자동안에 할수 있다면 더하기를 수행하는데 걸리는 시간은 ns로 얼마인가?

1.2 RAM과 ROM의 차이점을 설명하시오.

1.3 한 콤퓨터가 16MB의 기억용량을 가진다. 주소를 표시하는데 몇비트가 필요한가?

1.4 콤퓨터가 640byte의 기억기를 가진다. 그것은 정확히 몇bit인가?

1.5 일반적으로 CD-ROM은 몇byte의 정보를 가지는가?

1.6 학교에 있는 콤퓨터서고에서 장치에 대하여 모든것을 다 찾으시오. 최소한 다음의 정보를 얻어야 한다.

1) 극소형처리소자를 만든 회사이름
2) 처리소자의 동기속도
3) 장치에서 RAM의 크기
4) 하드디스크의 크기
5) 도형표시장치의 해상도
6) 조작체계의 이름과 준위

1.7 콤퓨터를 파는 한 회사의 광고를 찾으시오. 광고에서 첫 글자를 모아 만든 단어와 용어가 무엇을 의미하는지 찾으시오. 일부 용어들과 첫 글자를 모아 만든 단어들은 고속완충기억기, P&P, SCSI,

EIDE, EDO, EPP와 56K이다.

1.8 하드디스크의 용량은 3년에 한번씩 남은 가격의 두배로 늘어 난다. 현재 3.5인치하드구동기는 거의 30GB의 용량을 가지며, 거의 450딸라의 가격을 가진다. 6년후에 1TB디스크구동기를 구입하는데 돈이 얼마나 드는가?

1.9 사용하고 있는 조작체계에서 파일조작명령을 불러 보시오. 특히 다음의 동작을 수행하는 명령을 불러 보시오.

1) 파일지우기
2) 파일이름바꾸기
3) 파일복사
4) 파일이동
5) 등록부의 창조
6) 등록부삭제

1.10 다음의 수를 10진수값으로 구하시오.

1) 1001_2
2) 0374_8
3) 0110100_2
4) 4033_8
5) $A32E_{16}$
6) 2345_8
7) 1211_4
8) 0111111_2
9) $02F3D_{16}$
10) 01010010010011_2
11) 1776_8
12) $ABBA_{16}$
13) $ACDC_{16}$

1.11 다음의 수를 정의된 진수로 변환하시오.

1) 777_8을 16진수로
2) $AD11_{16}$을 2진수로
3) 01001011_2을 8진수로
4) 1111_{16}을 8진수로
5) 01001111_2을 16진수로
6) 01001111_2을 8진수로
7) 3771_8을 2진수로
8) 4356_{16}을 8진수로

1.12 다음의 8bit값 즉 2의 보수에 대한 10진수값을 구하시오.

1) 10111000_2
2) 10000001_2
3) 11000000_2
4) 10100101_2
5) 11111111_2
6) 10000000_2

1.13 다음의 합과 결과를 계산하시오. 2진수로 대답해야 한다.

1) 01000110+0001010
2) 00111011+0101100
3) 00000111+0000001
4) 00100111+0001111
5) 00010101-0001000
6) 00001000-0000011
7) 00001001-0000101
8) 00001011-0000100

1.14 다음의 질문과 관련하여 콤퓨터학자와 담화해 보시오. 담화를 2페지 쓰시오.

1) 왜 콤퓨터과학자가 되려고 하였는가?
2) 연구기술분야는 무엇인가?
3) 연구부문에서 가장 중요한 문제는 무엇인가?
4) 콤퓨터과학분야에서 가장 중요한 문제는 무엇인가?
5) 연구를 산업적으로 하는가? 산업대상은 어느 회사인가? 산업대상과 연구하는데서 유리한 점, 불리한 점은 무엇인가?

1.15 은행에서 사용하는 자동현금출납기(ATM)를 생각해 보시오. 다음의 사람들의 ATM에 대한 관련속성은 무엇인가 ?

1) ATM사용자
2) ATM수리공
3) 은행계산원
4) 은행사장

1.16 전화응답장치에서의 교갑화실례를 드시오.

1.17 대부분의 기관은 체계구조를 가진다. 체계를 증시하는 도표를 작성하시오.

1.18 다음의것은 객체인가? 자기의 대답을 증명하시오.

1) 아름다움
2) 시간
3) 수력
4) 나무
5) 질투

1.19 대부분의 전자장치는 모듈화원리에 의하여 설계되였는데 그것은 장치를 더 쉽게 만들고 수리할수 있게 해준다. 다음의 장치들의 기본구성요소나 모듈을 불러 보시오.

1) 텔레비죤
2) VCR
3) 초단파로
4) 전화기
5) 자전거
6) 라지오

1.20 객체지향계승체계에서 매 수준의 객체는 높은 수준의 객체보다 더 자세히 설명된다. 이 속성을 리용하여 체계의 실지실례를 세가지 드시오.

1.21 벌레잡이유희를 위한 창문클라스의 설계를 설명하시오. 클라스가 가지게 되는 속성 (즉 자료성원들) 과 동작(즉 성원함수들)이 주어 진다.

1.22 벌레잡이유희를 위한 마우스클라스의 설계를 설명하시오. 클라스가 가지게 되는 속성(즉 자료성원들)이 주어 진다.

1.23 벌레잡이유희를 위한 유희조종자의 설계를 설명하시오. 클라스가 가지게 되는 속성(즉 자료성원들)과 동작(즉 성원함수들)이 주어 진다.

1.24 련습문제 1.21부터 1.23까지의 설계를 리용하여 유희조종자객체, 벌레, 마우스, 창문사이의 호상작용을 보여 주는 도표를 그리시오.

1.25 카드유희의 객체지향설계를 설명하시오. 기본객체는 무엇인가. 이 객체들의 속성과 동작은 무엇인가. 객체호상작용은 어떤가?

1.26 일반장치조종에서 체계의 객체지향설계를 설명하시오. 기본객체는 무엇인가. 이 객체들의 속성과 동작은 무엇인가. 객체호상작용은 어떤가?

1.27 그림 1-22에서 계승실례는 여러가지 세부를 무시하였다. 다음의 문제점을 고려하시오.

1) 위치는 모든 벌레들의 한가지 속성이다. 벌레의 위치는 어떻게 정의되는가?
2) 죽이기통보를 받았을 때 벌레가 하게 되는 동작을 대충 이야기하시오.

1.28 Warp벌레클라스를 포함시켜 그림 1-22의 클라스체계를 확장하시오. Warp벌레는 때때로 없어 지고 새 위치에 나타난다. 벌레의 속성에서 어떤 변화를 일으켜야 하는가?

제 2 장. C++기초

소개

이 장에서는 여러개의 작은 C++프로그람들을 시험적으로 작성하고 C++에 의하여 제공되는 기본객체들을 소개한다. 목적은 C++프로그람의 일반적인 구조에 대한 표상을 주며 C++에 의하여 제공된 기본객체들에 익숙되도록 하는것이다.

흔히 하나의 프로그람작성언어를 구별하는 특징은 그 언어에 의하여 제공되는 기본객체들에 관계된다. C++는 기본객체들을 많이 가지고 있으며 그것들은 옹근수, 실수, 문자를 리용하여 만들어 지고 조작된다.

기본개념

- 함수 main()
- include
- 글자와 기호
- 정의
- 간단한 입출력대화
- 옹근수, 류점수와 문자형
- C++이름작성
- 선언
- 식
- 일반 2 진변환
- 연산자우선권
- 연산자결합
- iostream 출력과 입력

2.1 프로그람구성

대부분의 프로그람작성언어들은 실행단위를 가진다. 실행단위는 이름을 가진 프로그람명령문들의 모임이다. 하나의 프로그람은 이 실행단위들의 집합체로 구성된다. FORTRAN 이나 BASIC 와 같은 언어에서 단위를 부분루틴 혹은 부분프로그람이라고 한다. 다른 언어에서는 수속이라고 한다.

C++에서 실행단위는 함수이다. 이 C++실행단위는 1 개 파일 혹은 여러개의 파일안에 있을수 있다. C++코드를 포함한 1 개 파일을 콤파일단위라고 한다.

프로그람명령문들과 함수들을 하나의 묶음으로 만드는것은 많은 우점을 가진다.

첫째로 프로그람작성자가 고정적인것을 수행하는 작은 단위와 정의된 기능으로 코드를 구조화할수 있다. 이 구조는 프로그람의 복잡성을 줄일수 있다. 복잡성의 감소는 프로그람을 리해하기 쉽게 하고 수정하기 쉽게 하며 더 정확히 실행할수 있게 한다.

프로그람을 콤파일부분과 함수를 단위로 묶는것은 여러개의 절(실행단위)로 구성된 여러개의 장(콤파일단위)을 가진 하나의 책을 만드는것과 같다. 훌륭한 구성은 독자(프로그람작성자)가 책(프로그람)을 보기 쉽게 해준다. 함수를 리용하면 프로그람의 크기를 줄일수 있다.

프로그람에서 특수한 기능은 실행기간 여러 시점에서 수행된다. 례를 들어 프로그람은 여러가지 기능을 수행하고 어떻게 처리하여야 하는가를 사용자에게 자주 문의한다.

사용자는 대체로 Yes 혹은 No 로 대답한다. 그러한 질문은 프로그람을 구성하고 있는 함수들의 수

만큼 제기되며 응답을 받는 명령문들의 실행에 의하여 감소된다. 입력이 요구될 때마다 프로그람작성자는 함수만을 호출한다. 이 장에서는 하나의 함수를 가진 간단한 C++프로그람들을 실행한다.

2.2 첫번째 프로그람

기존방식대로 시험하려는 첫번째 프로그람은 다음의 통보를 출력하는 함수를 가진다.

Hello, world!

프로그람 2-1 에 이 프로그람의 원천코드를 주었다. 이 프로그람을 구체적으로 보자. 프로그람의 처음 3 개 행은 설명문이다. 설명문은 2 개의 빗선으로 시작된다. 설명문은 콤파일러에 의하여 실행코드로 콤파일되지 않고 설명하기 위하여 삽입되여 있으며 프로그람의 조작을 설명한다. 다음의 프로그람들은 너무 간단하여 설명하지 않아도 된다.

그러나 프로그람작성자들이 설명문을 쓰는것은 효과적이다. 후에 다른 프로그람작성자들이 프로그람에 대하여 의문을 가지면 설명문을 보고 그에 대하여 알수 있다. 2.4 에서는 이 프로그람을 더 많이 설명하였다.

```
// 프로그람 2-1: 인사말 표시
// 날자: 1/25/1998
#include <iostream>
#include <string>
using namespace std ;
int main() {
    cout<< "Hello world!" << endl;
    return 0 ;
}
```

프로그람 2-1. Hello world 프로그람

프로그람의 4~6 행

```
#include <iostream>
#include <string>
using namespace std;
```

은 거의 모든 프로그람에서 시작하게 되는 행들이다. 이 행들은 입출력을 진행하기 위하여 iostream 서고를 리용하는 프로그람들에서의 시작을 의미한다. 4, 5 행은 전처리명령들이다. 전처리기는 콤파일러전에 실행되는 프로그람이다. 사명은 콤파일러가 어느것을 원천코드로 보는가를 결정하는것이다. include 명령은 프로그람에서 정의된 파일의 내용과 명령을 바꾼다.

두 파일 iostream 과 string 은 프로그람이 사용할수 있는 출력기능을 가지고 있다. 파일이름을 막고 있는 왼쪽과 오른쪽각괄호는 이 파일들이 특수한 체계등록부에서 찾을수 있는 체계파일이라는것을 의미한다. 프로그람의 6 행

```
using namespace std;
```

은 std 라는 객체를 사용한다는것을 의미한다. 이 특수한 부분은 프로그람을 작성할 때 정의되는 많은 객체이름을 가지고 있다. 프로그람의 7행은 함수를 정의하고 함수가 돌려 주는 결과의 형을 지적한다.

C++프로그람에서 main 으로 정의된 함수는 프로그람이 콤파일되고 실행될 때 호출되는 첫 함수이다. 함수이름다음의 괄호들은 함수에 인수를 주는데 사용된다. 이 프로그람에서 함수 main()은 인수를 요구하지 않으며 괄호사이에 아무것도 쓰지 않는다. main 전에 쓰는 단어 **int** 는 main()이 돌려 주는 결과의 형을 지적한다. 단어 **int** 는 C++에서 옹근수형이름이다. 정의에 의하여 main()은 항상 옹근수결과를 돌려 준다. 다음의 괄호는 왼쪽대괄호 {이다. 괄호와 같이 대괄호는 무엇인가를 묶는데 리용된다. 여기서 왼쪽대괄호와 함수의 마지막에 있는 오른쪽대괄호는 함수를 이루는 프로그람명령문들을 묶는다.

이 함수는 2개의 명령문을 포함한다.

```
cout << "Hello world!" << end1;
return 0;
```

첫 명령문은 실지로 a+b+c 와 같은 형식이다. 여기서 cout 는 객체이다. 여기에서 조작에 대한 정의는 이름공간 std 와 iostream 파일안에서 찾을수 있다. iostream 에 서술된 객체들은 iostream 서고부분이며 입출력을 위하여 리용된다. cout 는 출력명령이며 대체로 현시장치에 관계된다. 두번째 명령은 문자렬이다. 문자명령은 인용괄호로 막는다. C++술어에서 <<연산자는 정의한 명령에 문자렬을 추가한다는 것을 의미한다. 결과는 문자렬 Hello world!를 현시장치에 표시하는것이다. 세번째 연산자 end1 도 iostream 서고의 한 부분이다. 이것을 조종자라고 한다. 조종자는 여러가지 특수한 동작을 일으키는 명령에 추가된다. 조종자 end1 은 출력명령에 행바꾸기를 진행하고(새행에서 다음출력을 시작하기 위하여) 화면에 직접 출력하기 위하여 현시장치로 보내는 모든 출력에 초점을 둔다.

프로그람의 마지막 두번째 명령은

```
return 0;
```

이다. 이 명령은 함수 main()을 끝내고 조종권을 조작체계에 넘긴다. 복귀명령에서 0 은 main 에 의해 돌려 지는 값이다. main 에서 결과 0 은 프로그람이 성과적으로 동작하고 오유가 없다는것을 의미한다. 0 이 아닌 결과는 여러가지 오유가 생긴것을 나타내며 프로그람이나 조작체계를 호출하여 해당한 동작을 할수 있다.

2.3 두번째 프로그람

프로그람 2-1 에서는 여러가지 C++구성요소들을 소개하였다.

프로그람 2-2 에서는 구입세액을 받아 그에 대한 판매세액을 계산한다. 다음의 행들은 프로그람의 실례이다.

```
Purchase Price $55.50
sales tax on $55.50 is $2.22
```

밑선 친 부분은 사용자가 입력한 질문에 대한 응답을 보여 준다. 이 책의 앞부분에서는 사용자의 입력응답을 이렇게 표시하겠다. 첫 실행명령문

```
cout << "purchase Price ?" ;
```

은 사용자가 구입세액을 입력할것을 지시한다. 이러한 명령은 사용자와 대화하는 프로그람에서 흔히 쓴

다. 명령에서처럼 같은 행에 유표가 있게 하자면 end1 이 사용되지 말아야 한다. 구입세액을 읽기 위하여 값을 기억해 둘 곳이 있어야 한다.

프로그람에서 명령문

```
float Price ;
```

은 C++콤파일러가 류점수값(즉 상수)을 가지는 Price 라는 이름을 가진 객체를 만들기 위한 정의이다.

```
// 날자:4/25/1998
#include <iostream>
#include <string>
using namespace std;
int main() {
    // 입력값
    cout << "Purchase Price ?";
    float Price;
    cin >> Price;
    // 판매세액의 계산과 출력
    cout << "Sale tax on $" << Price << "is" ;
    cout << "$" << Price * 0.04 << endl;
    return 0;
  }
```

프로그람 2-2. 구입세액프로그람

류점수객체에서는 세액이 입력되여 10 진수처럼 리용된다. 명령문

```
cin >> Price ;
```

는 건반으로 입력하는 수를 기다린다. 값이 정해 지면 이것은 내부형식을 류점수로 바꾸고 객체 Price 에 기억한다.

첫번째 연산수 cin 은 객체이다. cout 처럼 조작에 대한 정의는 파일 iostream 안에 있다. cin 은 입력명령객체이며 건반과 관련된다. 연산자 >>는 인수연산자라고 하는데 이것은 지정된 명령문으로부터 값을 얻는다는것을 의미한다. 얻은 값은 오른쪽연산수에 기억된다. 기본효과는 건반으로 지정한 수를 읽고 Price 에 기억하는것이다. 마지막 2 개의 명령문

```
cout << "sales tax on $" << Price << "is";
cout << "$" << Price * 0.04 << end1;
```

들은 현시장치에 결과를 표시한다. 첫번째 명령문은 문자렬을 표시하며 값은 Price 에 기억된다. 두번째 명령문은 계산 Price*0.04 의 결과를 출력하는데 그것은 판매세액이다.

그러므로 삽입연산자는 계산결과값을 출력할수 있다. 이 2 개 명령문들은 하나의 명령문과 같이 한 행에 쓰고 갈라 놓았다.

2.4 설명문

초기프로그람작성자들이 어려워 하는것은 프로그람이 콤퓨터에서 실행은 되지만 다른 사람들이 읽을 수 없는것이다. 그러므로 C++프로그람은 합리적이여야 하며(즉 콤퓨터로 리해할수 있어야 한다.) 다른 프로그람작성자들이 프로그람을 리해할수 있어야 한다.

수백명의 프로그람작성자들이 흔히 큰 판매용쏘프트웨어개발체계에서 일을 한다. 그중 일부 사람들은 다른 사람들이 오유를 수정하는 동안 새로운 기능을 추가한다. 프로그람작성자들이 새로운 기능을 추가하자면 프로그람의 동작을 리해하여야 한다.

따라서 다른 사람들이 리해할수 있도록 프로그람을 작성하는것은 중요하다. 사람들이 사용하는 프로그람을 작성하는 과정에는 수정이 반드시 있다. 프로그람을 작성할 때에는 그것이 어떻게 동작하는가가 리해되지만 2~3 달 또는 그이상 지난후에는 프로그람의 동작에 대하여 일부 중요한 내용들은 잊어 버린다. 설명문(comment)은 콤파일러에 의하여 처리되지 않는 문법이나 주소들이다. 주소는 프로그람의 동작방법을 설명한다.

C++에는 두가지 형식의 설명문이 있다. 첫번째 형식에서는 문자 '/'을 2 개 붙여 쓴다. 콤파일러는 //과 그 행의 모든것을 무시한다. 아래의

```
// comment
```

는 완전한 C++설명문이며 다음의 명령문은 옳은 설명문이 아니다.

```
// Not a comment due to the space between the /'s
```

프로그람을 식별하고 프로그람을 작성한 사람을 알려 주며 작성자를 기록하기 위하여 프로그람 2-1과 2-2 의 처음에 설명문을 사용하였다. 프로그람의 시작에 이 정보를 포함하는것은 표준규칙이다. 후에 다른 프로그람작성자가 프로그람에 대하여 질문이 생기면 그에 대하여 알수 있다. 다른 한가지 설명문은 프로그람을 변경시킨 수정자들을 추가하는것이다. 그 실례를 프로그람 2-3에 주었다.

```
// 날자: 4/25/1998
// 수정:
// 날자: 8/1/998
# include < iostream >
# include < string >
using namespace std ;
int main() {
    cout << "Hello world!" << endl ;
    cout << "Goodbye world!" << endl ;
    return 0 ;
}
```

프로그람 2-3. 수정한 Hello world 프로그람

C++설명문의 두번째 형태는 문자 /*로 시작되여 */로 끝나는것이다. 이러한 설명문은 하나이상의 행에 놓일수 있다. 콤파일러는 /*로 시작하여 */로 끝나는 사이의 모든것을 무시한다.

실례로 /*은 여러 행설명문이다. 이 설명문은 여러 행에 널려 있다.

```
/**************************************************************
 * Program 2-4 : Greetings variant                           *
 * Authors : James P. Cohoon and Jack W. Davidson            *
 * Date 4/25/1998 variant                                    *
 **************************************************************/
# include <iostream>
# include <string>
using namespace std ;
int main() {
    cout << "Hello world!" << endl ;
    cout << endl; /* output blank line */
    cout << "Bye world!" << endl ;
    return 0 ;
}
```

프로그램 2-4. /* … */형식의 설명문을 쓴 프로그램

일반적으로 C++프로그람작성자들은 //형식의 설명문을 사용한다.

두번째 형식은 오유수정목적으로부터 블로크코드를 일시적으로 소거할 때 리용한다. 소거된 블로크는 /* */로 둘러 막힌다. 이 형식의 설명문은 겹치지 않도록 주의하여 리용하여야 한다. 따라서 질문에서 코드블로크가 /* */형식의 설명문을 포함한다면 결과는 기대할수 없다. 프로그람 2-4 를 다시 보자. 마지막 2 개 명령문을 일시적설명문으로 소거하면 프로그람 2-5 가 얻어 진다.

```
/**********************************************************
 * Program 2-5 : Greetings variant                       *
 * Authors : James P. Cohoon and Jack W. Davidson        *
 * Date 4/25/1998                                        *
 **********************************************************/
# include <iostream>
# include <string>
using namespace std ;
int main() {
    cout << "Hello world!"<< endl ;
    /*
    cout << endl ; /* output blank line */
    cout << "Bye world!" << endl ;
    */
    return 0 ;
}
```

프로그람 2-5. /* … */블로크형식의 설명문을 리용한 프로그람

세번째 삽입명령문은 앞행의 마지막에서 */로 끝나므로 소거되지 않는다. 추가적으로 세번째 삽입명령(즉 */)의 다음행은 원천코드로 처리되지만 정확한 C++식은 아니다. 프로그람이 콤파일되면 콤파일러는 문법오유를 알려 준다.

문 제

1. 출력명령에서 행바꾸기는 어느 조종자에 의해서 진행되는가?
2. cout 명령으로부터 옹근수를 얻어 내는 C++명령문과 cout 객체에서 위치를 쓰시오.
3. 어느 상태에서 함수 main()이 0 아닌 값을 돌려 주는가?
4. 밀구입세액을 계산하는 C++프로그람을 작성하시오. 감모를 17%로 보시오.

2.5 값주기

세번째 프로그람은 앞부분에서 y 축자리표값을 계산한다. 프로그람에 대한 입력은 직선의 특성(즉 경사도와 y 축자리표)과 관계되는 x 축자리표값이다. 이것을 그림 2-1 에서 보여 주었다.

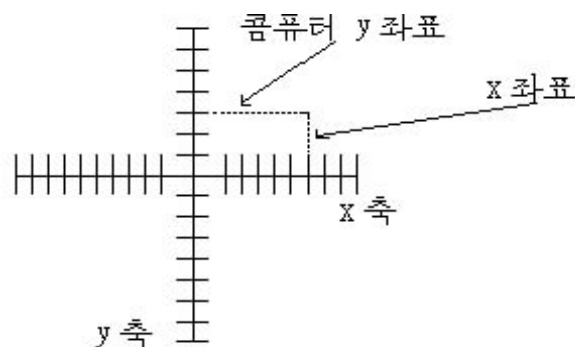



그림 2-1. 직선에 대한 평균

프로그람 2-6 에 그의 코드를 주었다. 이 프로그람은 앞에서 본 2 개보다 더 크지만 새로운 C++구조체인 값주기만을 사용한다. 첫 2~3 행에서는 사용자에게 알려 주고 직선의 경사도와 x 자리표값를 읽는다. 명령문

```
y= m * x + b;
```

는 y 축자리표값을 계산한다. 이 명령문은 y 가 m*x+b 와 같다는것이 아니라 y 객체에 m*x+b 의 계산결과를 넣는다는것을 의미한다. 값을 계산하고 기억하는것이 모든 계산에서의 기본원리이다.

```
# include < iostream >
# include < string >
using namespace std ;
int main() {
    //입력행파라메터
    cout << "slope of line (integer) ?" ;
```

```
    int m ; //선경사도
    cin >> m ;
    cout << "intercept of y-axis (integer)?" ;
    int b; //y 축자리표점
    cin b; //x 자리표점입력
    cout << "x-coordinate of interest (integer)?";
    int x; //y 자리표계산과 출력
    int y;
    y = m * x + b ;
    cout << "y = " << y << "when = " << m << ";" ;
    cout << "b ="<< b << " ; x = "<< x << endl ;
    return 0 ;
}
```

프로그람 2-6. 선우에서 y 축자리표계산

주의

잘된 형식의 프로그람

프로그람 2-6 은 훌륭한 프로그람작성실례이다. 프로그람은 무엇을 하는가 하는 간단한 설명으로 시작하며 프로그람작성자를 보여 주고 프로그람을 작성한 날자를 준다. 프로그람은 설명문을 포함하며 프로그람부분을 나타내는 빈 행도 사용한다. 일반적으로 알아야 할것은 프로그람의 시각적위치가 독자들이 프로그람의 물리적구조를 리해하는데 도움을 준다는것이다. 프로그람에서 저자들과 날자는 될수록 프로그람을 간단히 하기 위한것이다. 그러나 프로그람 2-6 에서 례증된 프로그람작성형식을 숙련하여야 하며 계속 장려하여야 한다.

2.6 C++의 기본객체들

C++는 기본객체형을 많이 가지고 있기때문에 위력하다. 그것은 대체로 하드웨어에 의하여 제공되는 모든 객체형에 접근하는 프로그람작성자들에게 있어서 좋은 수단이다.

기본객체형에는 3 개의 묶음 즉 옹근수객체형, 류점수객체형, 문자객체형이 있다. 매개 묶음은 여러가지 변수들을 가지고 있다. 다음부분에서 매 묶음과 일반적으로 사용되는 변수들을 보기로 하자. 다른 특수한 기본형변수들은 필요에 따라 책에 서술하였다.

2.6.1 옹근수객체형

대부분의 프로그람작성언어는 옹근수값을 기억하고 다루는 객체를 가진다. C++에서 기본옹근수객체형을 int 라고 한다. C++정의에서는 int 의 크기를 정하지 않았다. int 의 크기는 이상하게도 하드웨어와 콤파일러에 의존한다. 콤퓨터과학자들은 int 의 크기가 수단에 따라 정해 지며 콤파일러수정자가 자유롭게 int 형의 크기를 선택할수 있다고 하고 있다.

보통 int 는 하드웨어에서 가장 많이 쓰이는 옹근수자료형을 선택한다. 대부분의 PC 에서 int 의 크기는 16bit 이다.

C++는 여러개의 서로 다른 옹근수객체형을 제공한다. short(짧은)와 long(긴)이 있다. C++의 정의에서 short 와 long 객체형의 bit 들의 크기를 지정한다면 long 은 int 보다 짧지 않으며 int 는 short 보다 짧지 않다.

NumberOfBitsshort <=NumberOfBitsint <=NumberOfBitslong

short 는 **int** 보다 작으며 **long** 은 **int** 보다 크다는것이다. PC 에서 **short** 는 보통 8bit 이고 **long** 은 32bit 이다. UNIX 체계에서 **short** 는 대체로 16bit 이고 **long** 은 보통 32bit 아니면 64bit 이다. C++가 왜 세가지 서로 다른 옹근수객체형을 제공하는가 하는 질문이 생긴다. 실제로 다른데서도 잘 제공되지만 마지막까지 그에 대한 론의를 계속 하기로 하자.

오늘날 객체는 콤퓨터로 조종된다. 실례로 초단파발진기, 자동차, VCR, 립체음악재생기 등도 콤퓨터로 조종된다. 조작에서 콤퓨터가 중심역할을 하는 체계를 매몰체계라고 한다. 대부분의 매몰체계에서 체계를 조종하는 프로그람이 들어 있는 기억기가 제일 중요하다. 실례로 초단파발진기는 직접 발진기의 조작을 조종할수 있는 간단한 처리기를 가지고 있다. 발진기를 만드는데 관련되는것들중의 하나가 발진기의 조작을 조종하는 프로그람이 들어 있는 기억기소편수이다.

관계되는 객체의 종류와 규격이 주어 지면 프로그람작성자는 기억기에 있는 프로그람에 대한 요구회수를 줄일수 있다. 2～3 개정도의 기억기소편으로 설계한다면 발진기의 가격은 더 낮아 진다.

앞에서 언급한바와 같이 객체는 객체에 대한 속성 즉 값과 동작이나 조작의 모임이다. 옹근수객체형들인 **short**, **int**, **long** 에 대하여 C++는 더하기, 덜기, 곱하기, 나누기와 같은 일반산수연산을 제공한다. 또한 두 옹근수객체에 대한 비교연산자도 있다. 6 개의 비교연산자는 갈기, 안갈기, 작기, 작거나 갈기, 크기, 크거나 갈기이다.

2.6.2 문자객체형

명백히 옹근수객체형과 관련이 있는것은 문자객체형 **char** 이다. 문자들은 옹근수에 의하여 특정한 문자를 표시하는 여러가지 방법으로 코드화되여 있다. 실례로 옹근수 97 은 문자 a 를 표시한다. 사용된 코드화묶음을 문자모임이라고 한다.

오늘 사용하고 있는 2 개의 문자모임은 ASCII 와 EBCDIC 이다. 대부분의 콤퓨터들은 ASCII 문자모임을 사용하지만 EBCDIC 문자모임은 IBM 콤퓨터들에서만 사용되고 있다. ASCII 문자모임은 7bit 로 코드화되여 있고 EBCDIC 문자모임은 8bit 로 코드화되여 있다.

char 의 기정표현이 옹근수이므로 옹근수로 정의된 연산자는 문자형으로 정의된다. 사용하고 있는 문자모임대신에 다음의 관계가 주어 져 있다는것을 항상 생각하여야 한다.

'a'<'b'<'c'< … <'z'   'A'<'B'<'C'< … <'Z'

와

'0'<'1'<'2'<'3'< … <'9'

의 관계는 그것들이 자모순서로 배렬되여 있는 련속문자들로 된 때에 쓸모가 있다. ASCII 문자모임에서는 'a'+1 이 문자 'b'를 부호화한 옹근수를 만들고 'A'+1 이 문자 'B'를 부호화한 옹근수를 만든다. 즉 'z' 와 'Z'를 제외한 임의의 더 크거나 더 작은 문자 c 에 대하여 전반적으로 'c'+1 은 자모순에서 다음 문자를 만든다. 마찬가지로 식 '2'+1 은 문자 3 을 부호화한 옹근수를 만든다. 즉 '9'를 제외한 임의의 수 d 에 대하여 식 'd'+1 은 다음수자를 만든다. 이 관계는 그것이 더 작은 문자, 더 큰 문자를 표시하든지 수자를 표시하든지 명백하게 분류된 문자들에 대해서는 허용되므로 리용가치가 있다.

EBCDIC 문자모임에서는 앞에서 본 관계에 의하여 모든 문자들을 얻지 못한다. 실례로 EBCDIC 문자모임에서 식 'i'+1 은 문자 ';'를 코드화한 옹근수를 만들게 되며 문자 'j'는 아니다. 이러한 리유로해서 객체형 **char** 는 ASCII 문자모임에 의하여 정의한다.

제 4 장에서 보게 되는 렬거형과 함께 옹근수와 문자객체형은 옹근수형으로 알려 져 있는데 형모임을 만든다. 그것들은 옹근수의 2 진코드화에 의해 표시되므로 옹근수형이라고 한다.

2.6.3 류점수객체형

류점수객체형은 실수 즉 옹근수부과 소수부을 가지는 수를 표시한다. 례를 들어

3.1412

는 옹근수부 3 과 소수부 .1412 를 가진다. C++는 3 개의 류점수객체형 float, double, long double 을 제공한다.

C++에서는 int, short, long 이 지정되지만 float, double, long double 은 지정되지 않는다. 그 리유는 그것들이 하드웨어에 의존하기때문이다. 그러나 float 형으로 표시된 값은 double 형으로 표시된 값의 한 부분이며 double 형으로 표시된 값은 long double 형으로 표시된 값의 한 부분이라고 할수 있다.

류점수형식만을 제공하는 처리기에서는 float, double, long double 형이 같다. 2 개의 명백한 형식을 지원하는 처리기에서 float 형은 2 개 형식가운데서 더 작은것으로 되며 double 과 long double 은 더 큰 형식으로 될것이다. 한가지 실례로서 인텔의 x86 계렬에 속하는 개인용콤퓨터를 생각해 보자. 이 계렬은 2 개의 류점수표시를 지원한다. 하나는 single 형이다. 32bit 로 기억되며 7 자리 10 진수만한 정확도를 가지고

$1.18*10^{-38} \leqq x \leqq 3.40*10^{38}$

범위내의 수를 표시할수 있다. 두번째는 double 형실수이며 거의 15 자리 10 진수만한 정확도를 가지고

$2.23*10^{-380} \leqq x \leqq 1.80*10^{380}$

범위내의 수를 표시할수 있다.

double 형실수는 64bit 기억기를 요구한다. 이러한 구조에서 float 형은 single 형실수로 된다. double 형과 long double 형은 double 형실수로 된다.

C++는 비교연산뿐아니라 류점수자료형에 대한 일반산수연산을 제공한다. 류점수객체형의 리론적기초가 여러가지 크기를 가진 옹근수객체형에 대한 필요충분조건과 같다고 할수 있다. 더 큰 류점수형식은 더 큰 범위와 정확도로 수를 표시할수 있다. 이것은 응용프로그람을 최대로 만족시키는 형식을 선택하는 것과 같다.

2.7 상 수

앞부분에서는 C++기본객체형을 보았다. 이번에는 매형들에 대하여 상수를 어떻게 쓰는가를 본다. 즉 다른 프로그람작성언어들과 비교하면 C++에는 매형을 실수로 쓸수 있는 여러가지 방법이 있다. 문자렬상수를 먼저 보자.

2.7.1 문자렬과 문자상수

프로그람 2-1 에서는 문자렬상수

"Hello world !"

를 사용하였다. 문자렬상수는 인용괄호로 둘러 막힌 0 개이상의 문자렬이다. 이 간단한 수단에 특수한

문자들을 붙여 경보소리 혹은 형태변환과 같은 특수한 기능을 나타낼수 있어야 한다.

문자렬상수에 특수문자를 포함하는것을 C++에서는 의미해제(escaping)라고 한다. 확장문자렬(escape sequence)이라고 하는 특수한 문자는 그 다음문자의 의미를 변화시킨다. 례를 들어 C++에서 행바꾸기문자를 나타내는 확장문자렬은 역사선 '\'이 붙은 \n이며

"Hello world ! \n"

라고 쓴다. 역사선은 문자 n 이 n 그대로가 아니라 행바꾸기문자라는것을 나타낸다. 인용괄호를 포함하는 문자렬상수를 쓸수도 있다 .

즉 코드를 사용하여 "\"Hello, world ! "\"라고 쓴다.

명령문

```
cout << " \"Hello, world! "\" << end1;
```

은 현시장치에 문자렬(인용괄호를 포함하여)

"Hello, world !"

를 쓴다. 표 2-1 에서 C++확장문자렬의 목록을 주었다.

C++는 수값으로 주어 진 문자렬상수에서 정의된 문자들에 관하여 하나의 의미확장을 제공한다. 사용한 수는 8 진수 혹은 16 진수가 되여야 한다. 10 진수표기법은 허락되지 않는다. 이처럼 한 문자는 000 과 hh 가 8 진수 혹은 16 진수인

\o00 혹은 \xhh

형태가운데서 하나를 가질수 있다.례를 들어 문자렬상수

"Hello, world !\o12"

는 12 가 행바꾸기문자의 ASCII 부호이기때문에

"Hello world !\n"

과 꼭같은 문자렬이다.

표 2-1. 확장문자렬

특성이름	ASCII 값	C++확장문자렬
행바꾸기	NL	\n
수평타브	HT	\t
수직타브	VT	\v
공백	BS	\b
페지넘기기	FF	\f
경고, 삑소리	BEL	\a
자리복귀	CR	\r
역사선	\	\\
외인용부호	`	\`
겹인용부호	"	\"
물음표	?	\?

일반적으로 프로그람이 여러가지 문자모임으로 장치를 움직인다면 C++콤파일러는 특수한 문자를 위한 정확한 부호를 계산해 넣기때문에 C++문자가 사용하는데서 더 좋다. 문자렬상수가 기억기에 기억될 때 개별적인 문자들은 기억기의 련속적인 위치에 기억된다. 문자렬의 마지막문자의 다음에 0 문자('\0') 를 추가한다. 0 문자로 문자렬을 끝내는 약속은 문자렬의 끝을 쉽게 알도록 해준다. 또한 문자렬길이를 문자렬표시부로 기억할 필요가 없게 한다. 문자렬

"Hello world !"

의 기억기할당을 그림 2-2 에 주었다.

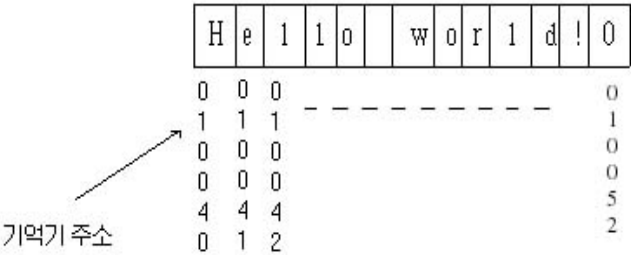



그림 2-2. 문자렬형문자의 기억기할당

매 문자는 1byte 의 기억기를 쓴다. 문자렬의 마지막 0 문자는 문자렬에서 문자의 수를 셀 때 써놓는다. 따라서 그우의 문자렬은 12 문자이다. 이 수를 문자렬의 크기라고 한다.

0 문자에 대한 문자렬상수 " "를 0 문자렬 혹은 빈 문자렬형이라고 한다. 그것은 0 문자 혹은 크기 0 을 포함한다. 그런데 0 문자는 마지막에 있으며 1byte 의 기억기를 쓰게 된다.

때때로 한 문자상수를 표시할 때가 있다. C++에서 한 문자상수는 '문자로 요구하는 한개의 문자를 둘러 막아 만든다. 실례로

'a' '1' '+' 'j'

은 정확한 C++문자상수이다. 명백한 식을 가지지 못하는 임의의 특수문자에 해당한 문자상수를 만들기 위해서는 문자렬상수에 대하여 정의한 의미확장기구를 쓸수 있다.

실례로 행바꾸기특수문자를 쓰기 위하여

'\n'

라고 쓴다. 다시 말하면 역사선은 문자 n 이 상수 n 으로 판단되지 않고 행바꾸기로 판단된다는것을 의미한다. 외인용부호를 가지는 문자상수에 대한 사용을 가정해 보자.

의미확장기구를 쓸수 있는데

'\'

라고 쓸수 있다. 외인용부호는 그속에 있는 문자가 정의된 문자라는것을 나타낸다.

또한 문자상수를 쓰기 위하여 의미확장수자렬을 사용할수도 있다. 문자상수는 000 hh 가 8 진수 혹은 16 진수인

'\000' 혹은 '\xhh'

형태들중 하나를 사용할수 있다. 다음의것은 8 진수형식을 사용하는 합리적인 C++ 문자상수들이다.

'\033' '\06' '\0177'

이것들은 각각 ASCII 문자인 ESC(탈퇴), ACK(확인응답) 그리고 DEL(지우기)을 표시한다.

수가 8 에 기초하고 있다는것을 표시하기 위하여 앞에 0 을 넣는것은 C++의 대표적인 형식이다. 다음의것은 같은 상수이다.

'\33'　　'\6'　　'\177'

문자들

'\x1b'　　'\x6'　　'\x7f'

는 16 진수형식이며 같은 문자들을 표시한다.

8 진수와 16 진수문자상수들에 관하여 콤파일러는 문자객체로 대응시키는데 너무 커서 수를 정의할수 없다면 오유를 알린다.

2.7.2 옹근수형상수

C++에서 옹근수형상수를 쓰는 가장 간단한 방법은 직접 수를 쓰는것이다. 실례로

23, 45, 101, 55

는 4 개의 정확한 C++옹근수형문자이다. 옹근수형상수를 쓰면 콤파일러는 C++객체형을 할당한다. 일반적으로 그것은 보통 int 형이지만 형은 상수의 크기와 뒤붙이에 의해 표시된다.

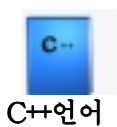
실례로 long 형처럼 처리되는 옹근수형상수를 쓰기 위하여 C++는 수의 마지막에 l 또는 L 을 추가할 수 있다. 따라서 상수

23 l　45L　101L　55L

은 모두 long 형이다. 작은 글자 l 은 수자 1 과 쉽게 혼돈되므로 얼마 쓰지 않는다.

short 형인 옹근수형상수를 정의하는 방법은 없다. 만일 옹근수형상수가 뒤붙이를 가지지 않는다면 그때 콤파일러는 값의 크기에 따라 형을 선택한다.

만일 값을 int 형으로 기억할수 있다면 그때 그 형은 int 이다. 그런데 값이 int 형으로 기억하기에 너무 크다면 콤파일러는 오유를 내보낸다. 여러가지 진수를 사용하여 옹근수형상수를 정의할수 있다. C++는 8 과 16 에 기초하여 옹근수형상수를 쓰게 한다.

C++언어

C++문법설명문

옹근수형상수를 만드는데는 여러가지 방법이 있다. 여러가지 실례를 보는것은 적당한 C++상수를 정의하는 일반적인 방법론을 주는데서 좋지만 C++구조체와 적당한 상수를 만드는 방법을 실례없이 설명하겠다. 여기서는 C++문법에 대하여 해설을 단 문법도표를 사용하여 서술하였다.

례를 들면 10 진수를 서술하는 문법도표는

1 이상의 수자렬 ;

이다. 여러개중 어느 하나를 정의하기 위하여 수직막대기 ('|')를 사용한다. 꺾쇠괄호는 임의로 지적하는 항목을 둘러 싼다. 따라서 형지적자는 L 아니면 l 일수 있으며 생략된다. 문법표에서 흘림체기호는 비말단이라고 한다. 즉 가능성모임을 표시한다. 표우에서 기호 Digits 는 수자 0, 1, 2, …, 9 를 표현한다. 설명문은 첫 수자가 0 이 아닌것을 가리킨다.

앞부분이 0 으로 시작하는 옹근수상수는 8 진수로 간주된다. C++상수

023　093　045　010

은 모두 8 진수에 기초하며 10 진수 19, 63, 37, 8 을 표현한다. 두번째를 제외한 모든것은 int 형이다. 두번째는 long 형이다. 상수가 8 진수라면 그때 문자 8 이나 9 는 상수로 나타날수 없다. 따라서 상수

038 093 0779

들은 정확한 C++상수가 아니다. 16 진수로 사용하기 위하여 앞붙이 0x 혹은 0X 를 사용한다. 16 진수상수에서 문자 a 부터 f 또는 A 부터 F 는 수 10 부터 15 를 표시한다. 문자

0x2a 0x450 xffL 0xAle

는 10 진수값 42, 69, 255, 2590 을 표현한다. 세번째 값은 long 형을 가지며 다른것들은 int 형을 가진다.

8 진수와 16 진수형상수를 위한 일반문법은

1 이상 수자렬 ;

이다. 프로그람 2-7 에 10 진수값을 8 진수, 10 진수, 16 진수상수로 출력하는 프로그람을 주었다.

```
//프로그람 2-7. Output different base constants
#include <iostream>
#include <string>
using namespace std ;
int main() {
    cout << "Display integer constants\n" << endl ;
    cout << "Octal constant o23 is" << o23 << "decimal"
      << endl ; // outputs decimal value 19
    cout << "Hexadecimal constant 23 is" << 23 << "decimal"
      << endl ; //outputs decimal value 23
    cout << "Hexadecimal constant 0x23 is" << 0x23
      << "decimal" << endl ; //Outputs decimal value 35
    return 0 ;
}
```

프로그람 2-7. 여러가지 진수에 대한 상수들의 출력

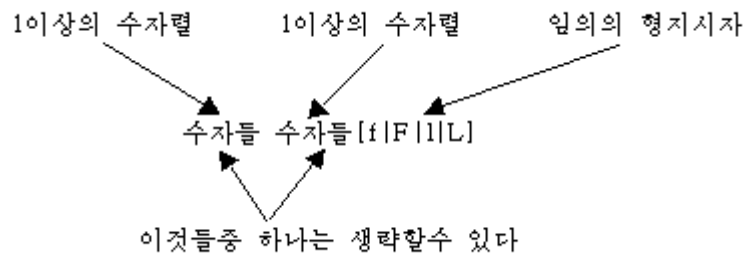
실행할 때 프로그람은 다음과 같이 출력한다.

Display integer constants
Octal constant o23 is 19 decimal

Decimal constant 23 is 23 decimal
Hexadecimal constant 0x23 is 35 decimal

수를 표시하는데서 여러가지 진수를 사용하는것은 합리적이지 못하다. 예민한 독자는 부수의 언급이 없다는것을 참고하여야 할것이다. 리유는 옹근수형상수가 항상 부수가 아니기때문이다. 부수로 만들기 위하여 미누스기호를 상수에 쓸수 있지만 형태적인 판단은 단항덜기연산자가 상수에 리용된것으로 된다. 미누스기호는 상수의 부분이 아니다. 그리고 플루스기호는 옹근수형상수에 리용된다. 그것은 상수의 값을 변화시키지 않는다. 플루스기호는 단항덜기연산자와 대칭되게 하기 위하여 C++에 포함시켰다.

2.7.3 류점수형상수

C++는 류점수형상수를 쓰기 위한 여러가지 방법을 제공한다. 류점수형상수의 한가지 형태의 문법은

이다. 실례로

2.34 3.1416 29.00 .23 0.32

는 모두 정확한 C++류점수형상수이다. 류점수형상수를 위하여 형을 다르게 정의하지 않는한 언제나 double이다.

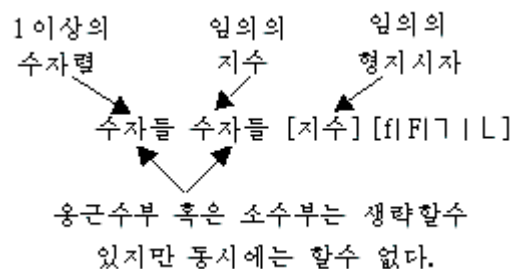
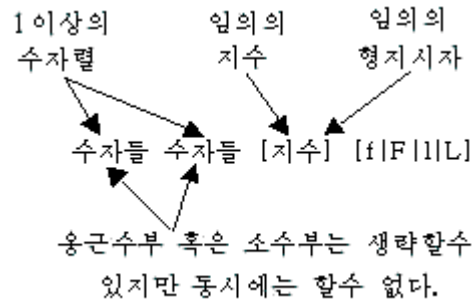
옹근수형상수를 리용하는것과 같은 방법으로 형은 문자 f, F, l, L 을 리용하여 뒤붙이로서 정의할 수 있다. 문자 f 혹은 F 는 상수가 float 형이라는것을 정의하며 문자 l 이나 L 은 상수가 long double 형이라는것을 정의한다. 류점수형상수의 형태들인

23.4f 0.21L 45.3F 7456.1

은 float, long double, float, long double 형이다.

C++는 또한 과학적표기법으로서 류점수형상수를 표시하는 능력을 제공한다. 이른바 과학적표시법으로 수를 10 의 제곱으로 표시한다. 수 $1.23X10^3$은 과학적표기법으로 된다. 이 수는 《1.23 에 10 의 3 제곱》이라고 읽는다. 우의 수자는 1230.0 과 같다. 과학적표기법에서 수의 일반적인 형태는

소수$\times 10^{지수}$

이다. C++과학적표기법을 위한 문법은

이며 지수는

이다. 소수는 옹근수 혹은 10 진수일수 있다. 지수는 부호 있는 옹근수이다. 과학적표기법을 리용한 C++ 류점수의 실례는

1.23E10 0.23E-4 45.e+23 23.68E12

이다. 이 상수들은 표준적인 과학적표기법으로서 각각 값

$1.23X10^{10}$ $0.23X10^{-4}$ $45.0X10^{23}$ $23.68X10^{12}$

으로 표현된다.

류점수형상수의 형태는 앞에서 언급한 뒤붙이를 리용하여 정의할수 있다. 우의 실례에서는 뒤붙이가 없기때문에 모든 수들이 **double** 형이다. C++상수

1.23E10F 0.23E-4f 45.e+23L 23.68E12L

들은 과학적표기법으로 표현된 우의 수와 같은 값을 가지지만 처음 2 개는 **float** 형이고 다음 2 개는 **long double** 형이다. 프로그람 2-8 에 여러가지 류점수형상수의 사용을 보여 주었다.

```
// 프로그람 2-8  Illustrate different forms of
// floating-point constants that have the same value
#include <iostream>
#include <string>
using namespace std ;
int main() {
    cout << 230.E+3 << endl ;
    cout << 230E3 << endl ;
    cout << 230000.0 << endl ;
    cout << 2.3e5 << endl ;
    cout << 0.23E6 << endl ;
    cout << .23e+6 << endl ;
    return 0 ;
}
```

프로그람 2-8. 같은 값으로 표현되는 류점수형상수의 서로 다른 출력형식

프로그람이 실행되면 다음과 같이 출력된다.

230000
230000
230000
230000
230000
230000

출력결과가 보여 주는것과 같이 모든 상수는 같은 값을 나타낸다. 2.7.2 에서 서술한 옹근수형문자상수인 경우와 같이 류점수형상수는 표시할수 없다. 상수는 덜기기호를 써서 반전시킬수 있다.

문 제

5. short 가 특수한 장치에서 16bit 라는것을 생각하시오. int 는 얼마만큼 커야 하는가 ?
6. 한 문자를 부호화하는데 사용하는 ASCII 문자는 몇 bit 인가 ?
7. 실수를 표현하는데 C++객체의 무슨 형을 리용하는가 ?
8. 행바꾸기를 비롯하여 확장문자렬이란 무엇인가?
9. 문자렬에서 ' '를 포함하여 렬을 확장하는것은 무엇인가?
10. 다음의 문자렬의 크기(즉 발생한 기억기의 바이트수)는 얼마인가?

    Ben Rush

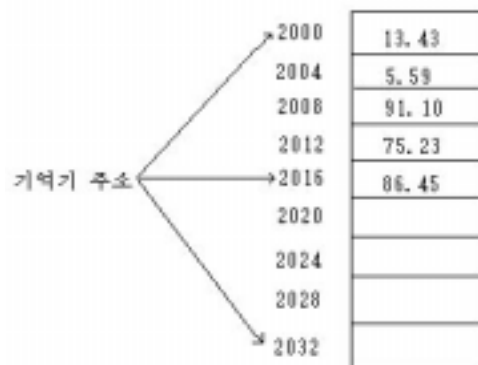
11. 2 개의 류점수구성요소를 지명하시오.
12. 류점수형상수 2.3E10 의 형은 무엇인가 ?
13. 류점수형상수 1.45E5L 의 형은 무엇인가?
14. C++류점수형상수 1.13E-10 을 과학적표기법으로 쓰시오 .

2.8 이 름

계산에서 기본요구는 정보를 기억하고 돌려 주는 능력이다. 아쎔블리어와 고수준언어전에는 프로그람작성자들이 기계어로 프로그람을 작성하였으며 값이 콤퓨터의 기억기 어디에 기억되여 있는가를 추적해야 하였다. 이 값들은 기억기위치주소를 지정하여 호출하였으며 값을 얻는다. 얼마나 지루한가를 나타내는 첫번째 측면과 처리하기 힘든 오유측면을 보기 위하여 5 개 항목에 대한 6%판매세액을 계산하는 간단한 기계어프로그람을 작성해 보자.

5 개 항목의 세액을 기억기에 기억한다. 그림 2-3 에 기억기의 위치를 주었다. 항목세액은 기억기의 위치 2000 부터 2016 에 기억되여 있다. 또한 항목의 세액과 계산세액의 합을 기억할 위치가 필요하다. 선택하는 위치가 항목의 세액이 기억된 기억위치와 일치하지 않도록 확보하여야 한다. 합과 판매세액을 주기 위하여 위치 2028 과 2024 에 제각기 정한다.

그림 2-3. 기계어프로그람실레에서 기억기배치

기억기위치는 n 이 호출하는 주소로 되는 M[n]을 써서 호출한다. 연산 M[n]=expr 는 기억기위치 M[n]에 expr 의 값을 넣는다. 판매세액을 계산하는 기계어

명령은

M[2028] ⟶ 0
M[2028] ⟶ M[2028]+M[2000]
M[2028] ⟶ M[2028]+M[2024]
M[2028] ⟶ M[2028]+M[2028]
M[2028] ⟶ M[2028]+M[2012]
M[2028] ⟶ M[2028]+M[2016]
M[2028] ⟶ M[2028]*.06

이다. 이 프로그람은 무엇을 계산하는지 말하기가 어렵다. 기계주소대신에 기호이름으로 명령을 다시 쓰면 오유가 덜하고 리해하기 훨씬 쉬운 프로그람이 얻어 진다.

Totalcost ⟶ 0
Totalcost ⟶ Totalcost + Price1
Totalcost ⟶ Totalcost + Price2
Totalcost ⟶ Totalcost + Price3
Totalcost ⟶ Totalcost + Price4
Totalcost ⟶ Totalcost + Price5
Salestax ⟶ Totalcost * 0.06

기호이름을 리용한 우점으로부터 모든 현대프로그람작성언어에서는 프로그람값이나 객체를 표현하는데서 이름을 사용한다. 모든 고수준프로그람작성언어에서 프로그람작성자는 다른 프로그람작성자가 프로그람내용뿐아니라 값을 의미하는 기호이름을 다르게 사용하도록 한다.

프로그람작성언어에서 중요한것은 정확한 이름을 만드는 규칙이다. C++에서 정확한 이름은 수자로 시작할수 없다는 추가적인 제한과 함께 문자(대문자와 소문자), 수자, 밑줄의 결합이다.

정확한 C++이름들의 실례는

x digit x1 score date AverageScore temp Nbr trials

이다. 정확한 C++이름이 아닌 문자렬의 실례는

2BORNOT2B TOH0t? $al A#

이다. 이름을 정확히 표기해야 한다. 즉 같은 이름에 대하여 문자부분을 포함하여 꼭 같은 글자를 써야 한다. 실례로

NbrOfTrials NbrofTrials

는 앞에서는 Of를 리용하였지만 두번째에서는 리용하지 않았으므로 다르다.

흥미 있는 질문으로서 이름의 길이를 얼마로 할수 있는가? C++의 정의에서는 이름의 길이에 대하여 제한하지 않는다. 그러나 일부 C++도구들은 이 점에서 결함이 있으며 그것들은 설정한 이름의 첫 n문자들만을 사용한다. 실례로 Turbo C++콤파일러는 이름의 첫 32개 문자를 사용한다. 다른 콤파일러들은 더 많이 혹은 더 적게 사용한다. 따라서 파일이름의 끝보다도 첫 부분에서 2개의 긴 이름이 갈다는것을 확인하는것은 중요하다.

다른 경우 프로그람작성자에게 서로 다르게 나타나는 2개의 이름이 콤파일러에 의하여 같은 이름처

럼 처리될수 있다. 수많은 다른 프로그람작성언어들과 같이 C++는 이름들에 대한 2 개의 클라스로서 열쇠단어(예약어)와 식별자를 가진다. 이 이름들은 다음절에서 본다.

2.8.1 예약어들

일부 단어들은 그 언어에 약속된것으로서 프로그람작성자가 이름으로 리용하지 못하는것도 있을수 있다. 이 특수한 이름을 흔히 예약어라고 한다 . 표 2-2 에 C++의 예약어들을 주었다.

예약어들은 콤파일러에 대하여 특수한 의미를 가지며 프로그람작성자는 변화시킬수 없다. 예약어를 이미 2~3 개 보았다. 예약어들인 **short**, **int**, **long**, **float**, **double**, **char** 는 C++의 기본형이다. C++로 더 깊이 연구하기 위하여 다른 예약어의 의미를 보기로 하자. 예약어는 소문자로만 구성된다는것을 명심해야 한다. 따라서 문자렬 continue DO char 는 예약어가 아니다. 그러나 다음절에서 보겠지만 그것들은 정확한 C++식별자이다.

2.8.2 식별자들

식별자는 프로그람작성자에 의하여 정의된 이름이며 의미가 있는것이다. 실례를 들어 프로그람 2-1 에서 main, cout, end1 은 기호이름들이다.

표 2-2. C++의 예약어들

Asm	else	operator	throw
auto	enum	protected private	true
bool	explicit	public	try
break	extern	private	typdef
case	false	register	typeid
catch	float	reinterpret_cast	union
char	for	return	unsigned
class	friend	short	using
const_cast	goto	signed	virtual
continue	if	sizeof	void
default	inline	static	volatile
delete	long	statinc_cast	wchar_t
do	mutable	switch	while
double	namespace	template	
dynamic_case	new	this	

main 은 함수이름이고 cout 는 현시장치로 출력할 때 사용하는 객체의 이름이며 end1 은 조종자의 이름이다. 정확한 식별자를 만드는 원리는 정확한 C++이름이여야 하며 예약어와 같아서는 안된다.

정확한 C++식별자의 실례는

TaxRate n Price flow first_value tmp

이다. 이름을 가진 객체의 목적을 함축하는 식별자를 선택하는것은 좋은 프로그람작성방법이다. 실례로 학급에서 학생수를 계산하고 기억하는 프로그람을 쓴다고 생각해 보자. 학급학생수를 넣기 위한 객체이름으로서 식별자 s를 선택할수 있지만 이 식별자는 내용이 매우 빈약하다. 즉

Number_of_students_in_class

를 사용하지만 이 식별자는 매번 쓰기가 힘들다. 보통 가운데부분이 합리적이다. 식별자 Nbrstudents 는 훨씬 더 짧고 거의 명백하다. 여기서는 객체들을 지명하기 위한 식별자들을 주고 만들기 위하여 여러가지 약속을 한다.

첫째로 한개 단어식별자를 사용하여야 한다. 그러나 객체의 목적을 명백하게 하기 위하여 둘이상의 단어로 된 식별자를 사용할 때는 매 단어의 첫 문자만을 리용하기로 한다. 실례로

Wordcount Time BitsPerSecond LapTime

과 같은 식별자를 사용한다.

둘째로 명백한 단어들은 흔히 략어를 사용한다. 실례로 Number를 Nbr, Object를 obj, count를 cnt로 하는 략어는 명확성을 잃음이 없이 식별자의 길이를 감소시킨다. 략어의 이러한 형식을 실례로 하는 식별자는

WindowObj EmployeeNbr WordCnt

이다. 프로그람을 개발하는데서 나타나게 될 추가적인 이름달기약속이 필요하므로 그것들을 소개한다.

알림

괄호의 사용

우선권의 원리를 아는것은 중요하지만 필요하지 않는 부분까지 명백한 평가순서를 만들기 위해 괄호의 사용을 제안한다. 이것은 프로그람작성자를 도와 주며 후에 코드가 변화되면 다른 프로그람작성자들이 오유를 일으키지 않도록 한다.

괄호 없는 식

a*b+c/d-3.0;

과 괄호 있는 식

(a*b)+(c/d)-3.0;

을 보자. 두 식은 정확히 같은 계산을 수행하지만 두번째의 의미가 훨씬 더 명백하다. 이 책에서 코드는 복잡한 식에서 명백한 평가순서를 만들기 위하여 늘 괄호를 사용한다.

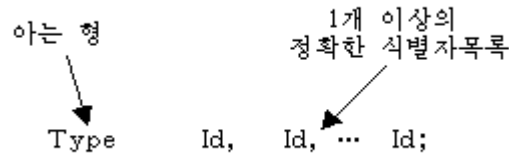
C++언어

밑선의 리용

밑선은 C++이름의 시작에 리용되지만 밑선으로 시작하는 식별자는 C++콤파일러에 의하여 처리되지 않는다. 이와 류사하게 하나의 밑선으로 시작하는 이름들은 조작체계루틴이름지정을 위한 일부 C기구에 의하여 처리되지 않는다. 일부 C도구들은 C서고를 리용한다.

2.9 정의

C++에서는 객체를 쓰기전에 객체를 정의하여야 한다. 프로그람에 객체의 이름을 소개하고 객체의 형을 지정한다.

C++정의의 일반형태는

이다. Type 는 기본형 혹은 프로그람작성자에 의하여 이전에 정의된 형이며 id 는 C++식별자이다. 정의 **int** sum; 은 초기값이 없는 **int** 형을 가진 sum 이라는 객체를 정의한다. 즉 객체는 지명되고 기억기는 할당되지만 객체는 주어 진 초기값은 없다. 정의의 이 보충적인 실례는

```
int X;
int WordCnt,  Radius,  Height;
float FlightTime ,  Mileage, Speed;
```

이다. 이 객체들에는 초기값이 주어 져 있지 않다. 일반적으로 초기화하지 않고 객체를 정의하는것은 좋지 않다. 실례로 Mileage 의 값을 명백히 주기전에 정의한다면 얻는 값은 모르거나 정의되여 있지 않다.

경험

객체란 무엇인가?

일부 책들에서는 객체와 기본자료형사이를 구별할수 있게 설명하고 있다. 객체란 자료가 하나의 단위로 결합되여 있으며 조작되는 자료와 함수가 있는 들어 있는 자료형의 한가지이다. 이 기본자료형에 대한 선언은 객체가 아니라 변수형이다. 이러한 구별은 인위적인것이다. 객체는 값을 가지는 기억기의 한 부분이다. 이 값들은 객체의 형에 따라 어떻게 판단되며 어떻게 호출되는가, 형이 프로그람작성자에 의하여 창조되는가 프로그람언어에 의하여 제공되는가 하는데는 무관계하다.

프로그람 2-9 에 초기화하지 않은 객체의 리용에 대하여 주었다. 프로그람은 4 개의 초기화하지 않은 객체를 정의하고 그다음 그 값들을 출력한다. 프로그람이 동작할 때 출력은 예언할수 없다. 다음의것은 프로그람이 실행되여 출력된것이다.

```
f's value is 1.81825e+11
i's value is 8653
c's value is e
d's value is 1.12975e-231
```

따라서 정의할 때 객체를 초기화하는것이 좋다. 정의의 또 다른 형태는 정의될 때 초기화되는 객체를 허락하는것이다. 이러한 형태의 정의문법은

이다.정의 **int** sum=0;은 첫번째 정의와 같지만 sum 을 0 으로 초기화한다. 이러한 형태의 정의의 다른 실례들은

```
float  TaxRote=0.06;
```

char letter='a';

이다. 첫번째 정의는 TaxRate 라는 float 형객체를 0.06 으로 초기화하여 만들고 두번째는 char 형이며 초기값이 'a'인 letter 라는 객체를 만든다.

```
// 프로그람 2-9 : 초기화하지 않은 객체의 값출력
# include < iostream >
# include < string >
using namespace std ;
int main() {
    float f ;
    int i ;
    char c ;
    double d ;
    cout << "f's value is" << f << endl ;
    cout << "i's value is" << i << endl ;
    cout << "c's value is" << c << endl ;
    cout << "d's value is" << d << endl ;
    return 0 ;
}
```

프로그람 2-9. 초기화하지 않은 객체의 값출력

흔히 객체가 무엇을 표현하는가를 설명하기 위하여 정의와 함께 결합한 설명문을 사용한다. 다른 실례

```
int Temp=32 ;
char c ;
float PayRate ;
```

에서 객체가 하나의 계산일 때 설명문에서 계산단위를 나타내는데 도움을 주고 있다는것을 명심해 두는것이 좋다. 여러개의 객체는 하나의 명령문으로 정의하고 초기화한다. 실례로 C++서술

```
int count=0, Bits=16, small=-1 ;
```

은 모두 int 형인 3 개의 객체들을 서술하였는데 count 는 0 으로 초기화되고 Bits 는 16 으로 초기화되고 small 은 -1 로 초기화된다. 프로그람객체들의 기호이름을 사용하도록 한데 따라 정의의 사용은 콤파일러가 기억기에서 객체를 어디에 넣는가를 결정한다.

판매세액프로그람에서 여러가지 객체가 충돌하지 않거나 중복되지 않도록 기억기위치를 확보해야 한다. 객체에 기억기를 할당하는 이 과정을 기억기할당이라고 한다. 기억기를 수동적으로 할당하기 위하여 매 항목을 넣는데 요구되는 기억용량을 알아야 한다.

실례에서 세액을 류점수값으로 표현하는데 4byte 가 요구된다고 가정한다. 2～3 개의 객체들을 가진 프로그람을 위하여 프로그람작성자는 객체를 기억하는 위치를 추적할수 있다. 그러나 이 과제는 객체가 많은 큰 프로그람에 대해서는 힘들다. 따라서 정의는 콤파일러가 여러개 동작하도록 지시하여 객체를 넣

는 기억기위치를 할당한다. 이 기억기위치는 그때에만 보존되며 정의된 객체를 계속 넣어 사용할수 없다. 콤파일러는 이름이 련속 사용될 때 기억기에서 객체의 위치를 찾을수 있게 하기 위하여 이름과 함께 정보를 따라 기록한다. 그리고 마지막에는 객체와 초기값을 준다.

주의

항상 객체에 초기값을 준다.

일반적인 오유는 객체에 초기값을 주는것을 잊는것이다. 이것을 피하기 위하여 정의할 때 객체에 초기값을 주는것이 좋다. 제한조건은 입력명령으로부터 값을 골라 객체에 기억시켜 정의한 후 즉시 객체를 초기화한다는것이다. 실례로 코드

```
float Price;
cin >> Price;
```

는 가능하다. 그러나 실지로 쓰는데는 방해되지는 않는다.

```
float pirce=0.0;
cin >> Price;
```

그러나 일부 사람들은 객체 Price 가 쓴 즉시 값을 주므로 우와 같은 경우에는 혼란이 생길수 있다고 한다. 우의 코드는 조작을 통하여 객체에 직접 값주기를 하지 말고 항상 객체에 초기값을 주어야 한다는것을 보여 준다.

우에서 소개한 두가지 형태는 같이 사용할수 있다. 다음의 C++정의

```
int i,j =4, k, l=10;
```

는 4 개의 int 형객체 i, j, k, l 을 만든다. 객체 i 와 k 는 초기값이 없지만 j 와 l 은 각각 4 와 10 으로 초기화되였다. 여기서 기본원리는 객체를 초기화하는 정의를 사용하는것이다.

대체로 행당 하나의 객체만을 정의한다. 이 약속은 객체의 목적을 설명하는 매 정의와 설명문을 포함하여 할수 있다.

문 제

15. C++이름을 만들기 위한 적당한 문자는 무엇인가?
16. Window$cost 는 적당한 C++이름인가?
17. C++에서 특수한 의미를 가지는 이름을 무엇이라고 하는가
18. int 형객체 HeadCount 를 정의하고 25 로 초기화하는 C++정의를 쓰시오.
19. float 형객체 MovingAverage 를 정의하고 25.0 으로 초기화하는 C++정의를 쓰시오.

2.10 식

1 장에서 객체의 일반개념을 소개하였다. 객체란 속성이나 값 그리고 수행할수 있는 조작들의 모임이라는것을 상기하자.

식은 객체에 적용하는 조작을 위한 기구이다. 일반적으로 식은 낡은것으로부터 새 객체를 연산했다는것을 의미한다.객체나 연산값을 연산수라고 한다.

연산수에 연산을 적용하는 처리를 식평가라고 한다. 식의 평가는 값뿐아니라 형을 가지는 결과를 만든다. 식의 평가에서 값과 C++형을 만든다는 일반개념은 아주 중요하다. 이 개념을 매우 명백히 하기 위하여 식을 론의하는 다음절에서 2 개 조의 형태로 식을 평가한 결과를 보게 되는데 <value, type>

에서 value 는 표현값이고 type 는 값과 관련된 C++형이다.

2 개 조라는 말에 구애될 필요는 없다. 결과가 2 개의 구성요소로 된다는것은 바로 일반적인 방식이라는 뜻이다. 이 약속대로 표시한것을 상기시키는 의미에서 2 개 조로 표시한 결과를 경사체형식으로 표시하겠다. 마지막절에서는 표시법을 없애고 형을 기정으로 결과의 부분이라는것만 가정하여 값만 주겠다.

2.10.1 간단한 식

C++식에서 가장 간단한 형식은 리용한 연산이 없는 상수이다. 실례로 식평가

```
23 ;
```

은 결과 <23, int>를 만든다. 표시법의 반두점은 표시의 구분이나 끝을 나타내는 C++경계이다. 이와 같이 식

```
18.53 ;
```

은 <18.53, double>과 같다. 표시 'a' ; 는 <97, int>와 같다.

식은 또한 연산을 쓰지 않은 객체일수도 있다. 식의 이러한 형식을 평가한 결과는 객체의 값이다. 실례를 들어 다음의 서술과 식을 생각해 보자.

```
int XCoord;
XCoord;
```

식평가결과는 <23, int>이다. 여러가지 의미에서 연산은 연산자 Xcoord 에 맞는다. 적응되고 있는 연산은 X 에 coord 에 기억된 값을 읽어 내는것이다. 다음의 더 긴 실례를 보자.

```
double BattingAVg = .253;
int AtBat s = 301;
short stolenBases = 34;
float EarnedRunAvg =1.7;
char c = 'x';
AtBats;
BattingAvg;
EarnedRunAvg;
c;
stolenBases;
```

평가결과는 각각 <301, int>, <0.253, double>, <1.7, float>, <120, int>, <34, short>이다.

2.10.2 2 진연산조작

C++에는 옹근수와 류점수형에서 연산을 진행하는 여러개의 2 진연산수가 있다. 용어 2 진수는 연산자가 2 진연산에 적용된다는것을 의미한다. 2 진수조작을 수행하는 C++원리는 오히려 더 복잡하므로 2 진연산자에 옹근수형값을 써보자. 그다음 2 진연산자에 류점수도 포함하여 표시하자.

이 절은 옹근수와 류점수값을 비롯하여 C++에서 식을 어떻게 조종하는가 하는 론의로서 끝낸다.

2 진옹근수연산자들을 표 2-3 에 주었다. 모든 실례에서는 int 형을 사용한다. 표에서 볼수 있는바와 같이 2 진옹근수형산수연산자는 거의 대부분 할수 있다. 간단한 식 2+3 ; 은 값 <5, int>를 만든다. 이와

류사하게 식 4-7 ; 은 값 <3, int>를 만든다. 그러나 나누기와 나머지연산자는 특별히 주목을 해야 한다.

식 6/4 ; 는 값 1.5 가 아니라 1 을 만든다는것을 명심하여야 한다. 2 개의 옹근수값에 적용한다면 C++나누기연산자는 옹근수결과를 만든다. 만일 나누는 수가 나누일 수를 완전히 나누었다면 상의 소수부는 버리고 상의 전체를 결과로 한다. 소수부를 없애는 과정을 자르기라고 한다.

두번째 나누기시험에서 식 11/4 ; 는 결과 <2, int>로 평가된다.

표 2-3. 2 진옹근수상수연산자들

연산	연산자	실례	결과
더하기	+	2+3;	<5, int>
		5+10;	<15, int>
덜기	-	13-4;	<9, int>
		4-7;	<-3, int>
곱하기	*	3*4;	<12, int>
		5*11;	<55, int>
나누기	/	8/2;	<4, int>
		6/4;	<1, int>
		11/4;	<2, int>
		4/5;	<0, int>
		6/0;	<ϕ, int>
나머지	%	10%3;	<1, int>
		23%4;	<3, int>
		5%0;	<ϕ, int>

즉 나누기결과 2.75 는 값 2 를 만들기 위하여 자른다. 자르기는 반올림과 같지 않다는것을 참고하시오. 이 상태에서 반올림하면 값은 3 이 생긴다. 연산수가운데 하나가 부정확하고 결과가 부정확하다면 어떻게 되는가? 이 경우 C++의 정의는 2 가지 선택중의 하나이다.

2 가지선택은 대수적인 상으로서 가장 명백한 옹근수들이다. 콤파일러기구는 목적기구를 위하여 가장 편리한 결과를 선택하는데서 자유롭다. 실례로 식의 구성요소값의 선택 -11/2 ; 은 -5 와 -6 이다. 이렇게 콤파일러와 장치에 따라 결과는 <-5, int>와 <-6, int>가 된다.

마지막에 중요한 문제가 있다. 나누는 수가 0 이면 나누기연산결과는 없다. 그러므로 대부분 장치에서 0 으로의 나누기는 프로그람을 오유로 보고 정지시킨다. 이 책의 많은 실례에서 0 으로의 나누기가 우연히 수행되지 않도록 하는데 특별히 주의를 돌리였다. 나누기에 명백하게 관련되는것은 모드연산자 %인데 나누기의 나머지를 결과로 한다. 흔히 모드연산이라고 한다. 식 19%5 의 결과는 19 를 5 로 나눈 상이 3, 나머지가 4 로 되기때문에 ＜4, int＞이다.

모드연산자는 대체로 목적장치와 나누기지령을 사용하여 수행되기때문에 모드연산자에서는 많은 성질들이 있다.

첫째로 오른쪽연산수가 0 이면 연산결과는 없다.

둘째로 연산수가 부수가 아니면서 결과가 0 이 아니면(즉 나머지가 있다.) 결과값요소는 나누기를 수행하는 목적장치의 동작에 의존한다.

항상 식 (a/b)*b+a%b 는 b 가 0 이 아니면 a 와 같은 경우이다. 실례로

7/-2

은 결과 ＜-4, int＞를 만들며 식

7%__2

는 <-1, int＞를 만들어야 한다. 다른 속성에서 만일 식

7/-2

이 결과 <-3, int>를 만들면 나머지연산자는 결과 <1, int＞를 만들어야 한다. 모든 2 진옹근수형산수연산자는 기본장치가 조종할수 있는것보다 더 큰 값을 만들수 있다. 이 경우를 넘침이라고 한다.

옹근수형산수연산이 넘침을 일으키면 연산에 의하여 만들어 진 값은 없으며 프로그람의 동작은 예견할수 없다. 만일 2.6.1 을 상기한다면 C++는 char 형뿐아니라 3 가지 옹근수형 short, int, long 을 가질것이다. 이것과 함께 연산수는 모두 int 형이라고 간주한다. 다른 옹근수형에서 산수연산은 어떻게 수행되는가? 실례로 두 long 형값의 더하기는 두 int 형값의 더하기와 다른가? 만일 한 값이 int 형이고 다른것은 long 형이라면 어떻게 되겠는가?

4 가지 형태로부터 10 개의 더하기가능성이 있다. 조종하여야 할 부분의 수를 줄이자면 C++는 연산을 수행하기전에 연산수에 맞는 변화모임을 정의한다. 이 변환을 일반단항변환이라고 한다. 일반단항변환은 char 형과 short 형값이 연산을 수행하기전에 int 형으로 변환된다는것을 의미한다. 이 점에서 옹근수형 2 진연산의 연산수는 int 형이 아니면 long 형이다. 추가적으로 변환모임은 2 진연산이 적용되기전에 연산수에 적용된다. 이 변환을 일반 2 진변환이라고 한다. 만일 연산수가 같은 형이라면 변환이 없이 연산을 진행하며 결과형은 연산수의형 그대로이다. 연산수형이 같지 않다면 그때 int 형은 long 형으로 변환되고 long 연산을 수행하며 결과형은 long 이다. 이것은 복잡한 bit 로 보이지만 실지로는 거의 어렵지 않다. 이때 결과는 long 형이며 한개의 연산수가 long 형이 아니라면 결과는 항상 int 형이다. 이 원리를 표 2-4 에 주었다.

표 2-4. 왼쪽과 오른쪽연산수사이의 관계

왼쪽연산수의 형 \ 오른쪽연산수의 형	char	short	int	long
char	int	int	int	long
short	int	int	int	long
int	int	int	int	long
long	long	long	long	long

표 2-5. 2 진류점수산수연산자들

연산	연산자	실례	결과
더하기	+	2.0+.33;	<2.33, double>
		5.1+10.0;	<15.1, double>
덜기	-	13.6-4.2;	<9.4, double>

연산	연산자	실례	결과
		4.0-7.0;	<-3.0, double>
곱하기	*	3.0*4.4;	<13.2, double>
		7.5*11.0;	<82.5, double>
나누기	/	8.6/2.0;	<4.3, double>
		5.0/4.0;	<1.25, double>
		-11.0/4.0;	<-2.75, double>
		6.0/0.0;	<φ, double>

모드연산자는 류점수연산수에 적용될 때 의미가 없다. C++번역에서는 류점수값의 모드연산이 성립되지 않기때문이다. 다른 2진류점수산수연산자는 기대한대로 동작한다.

2 진옹근수형산수연산자와 같은 때 넘침이 일어 날수 있으며 0 으로의 류점수나누기는 프로그람정지를 일으킨다.

2진옹근수형산수연산자와 마찬가지로 일반2진변환은 연산수가 서로 다른 류점수형일 때 적용된다.

원리는 옹근수형에서의 원리와 갈다. 즉 정확치 못한 연산수는 다른 연산수만큼 정확히 하기 위하여 변환된다. 그렇지 않으면 결과에서 반드시 정확성을 잃는것이다. 실례로 하나의 연산수가 **float** 형이고 다른것이 **double** 형이라면 **float** 형연산수는 **double** 형으로 변환되여야 한다.

수행된 연산은 정확한 **double** 형더하기이며 결과의 형은 **double** 이다.

서로 다른 연산수로써 2진류점수연산자의 결과를 얻을 가능성을 표 2-6에 주었다.

표 2-6. 2진류점수연산에 대한 결과형

왼쪽연산수의 형 \ 오른쪽연산수의 형	float	Double	long double
float	float	Double	long double
double	double	Double	long double
long double	long double	long double	long double

연산수가 서로 다른 류점수형일 때 일반2진변환을 리해하는데 도움을 주기 위하여 변환을 요구하는 코드부분을 보자.

```
float Temp = 23.3;
double Volum = 3.2;
long double AvogadroConstout = 6.023E 23;
cout << volume *AbogadroConstant;
cout << Temp/volum;
```

cout 에서 volume 은 AvogadroConstant 가 **long double** 이므로 **double** 로부터 **long double** 로 변환한다.

여기서 중요한 개념은 정확한 연산수와 같이 정확한 산수연산을 사용하여 연산을 진행하기 위하여 연산수를 변환시켜야 한다는것이다.

2.10.3 단항산수연산

C++는 여러가지 단항연산자를 가진다. 용어 단항은 연산자가 하나의 연산수에 적용되는것을 의미한다. 2.7.2와 2.7.3에서 언급한바와 같이 C++는 값의 변환을 위하여 단항덜기연산자를 가질수 있다.

식

```
-23 ;
```

은

```
0-23;
```

로서 판단되며 명백히 결과는 <-23, int>이다.

단항덜기연산자는 수값을 가지는 지정된 객체에 적용될수 있다. 실례로

```
int i=10;
float x=12.3;
long Time =33;
//begin expressions
-i;
-x;
-Time;
```

코드부분에서 마지막 3개 행은 단항덜기연산자를 객체 i, x, Time에 적용하는 식이다. 이 부분에서 객체의 값은 0부터 시작하여 결과를 만들 때까지 덜어 진다. 결과는 <-10, int>, <-12.3, float>, <-33, long>이다.

C++는 또한 단항더하기연산자를 가지는데 단항덜기연산자와 대칭으로 포함한다. 식

```
+244
```

는

```
0+244
```

로서 판단되며 결과는 <244, int>이다.

2.10.4 원의 면적

흔히 아는 개념을 례증하기 위하여 간단한 문제를 푸는 프로그람을 작성하자. 문제는 반경이 주어진 원의 면적과 원주를 계산하라는것이다.

프로그람의 입출력조작은

```
Circle radius(real number) ? 5.1
Area of circle with radius 5.1 is 81.7104
Circumferance is 32.0433
```

이다. 프로그람을 작성하기전에 하나의 본질적문제를 고려하여야 한다.

원의 면적은 πR^2이라는것을 상기하자. C++는 지수연산자를 가지지 않지만 이 연산은 자체로 반경을 반복적으로 곱하여 수행된다. 따라서 반경을 가지는 객체의 이름이 Radius라면 원의 면적은 식

3.1415 * Radius * Radius

에 의해 계산될수 있다. 사용자들이 환경과 같은 류점수를 입력하게 하기때문에 Raduis 는 C++류점수형이다. 문제에 반경의 가능한 크기나 결과의 정확성이 지정되여 있지 않으므로 Radius 를 위하여 float 형을 임의로 선택한다. 프로그람 2-10 에 문제를 풀기 위한 코드를 주었다.

```
// 프로그람 2-10 : 원의 면적과 원주를 계산
# include < iostream >
# include < string >
using namespace std ;
int main() {
    cout << "circle radius (real number)?" ;
    float Radius ;
    cin >> Radius ;
    cout << "area of circle with radius" << Radius
      << "is " << (3.1415 * 2 * Radius) <<endl ;
    cout << "circumference is" << 3.1415 * 2 *Radius
    return 0 ;
}
```

프로그람 2-10. 원의 면적과 원주를 계산

2.10.5 혼합식

지금까지 식이 옹근수값이나 류점수값을 포함할 때 진행되는 변환을 보았다.

혼합식은 옹근수와 류점수형을 둘다 포함한다. 실례로 식

23-13.2 ;

에서 왼쪽연산수는 int 형이고 오른쪽연산수는 double 형이다.

결과를 볼수 있게 하기 위하여 식을 같게 하는 방법을 알려 주는 원리가 요구된다. 여기서는 왼쪽연산수를 double 형으로 변환하고 정확한 double 형덜기를 수행하면 결과 <9.8, double>을 만든다.

일반적으로 연산수가 류점수인 2 진수표시에서 연산은 류점수산수연산을 리용하여 수행되며 결과는 C++류점수형중의 하나이다.

일반탄창부호변환은 늘 옹근수형연산수를 int 형으로 변환하므로 혼합형산수연산을 하기 위한 일반 2 진변환표에 2 개의 보충적인 행과 렬을 추가하여야 한다.

표 2-7 에 결과형을 주었다. 다음의 코드부분을 분석하시오.

```
int MyDebt = 150 ;
double NationalDebt = 3.5E9 ;
float interestRate = 0.06 ;
long uspopulation = 200000000 ;
MyDebt *interestRate ;
NationalDebt / uspopulation ;
```

MyDebt 를 계산하는 첫번째 식은 결과 <9, float>를 만든다. 매 사람당 채무를 계산하는 두번째 식은 결과 <17.5, double> 을 만든다.

표 2-7. 혼합식연산에서 결과형

왼쪽연산수의 형 \ 오른쪽연산수의 형	int	long	float	double	long double
int	int	long	float	double	long double
long	long	long	float	double	long double
float	float	float	float	double	long double
double	double	double	double	double	long double
long double	long double	long double	long double	long double	long double

2.10.6 우선권

많은 프로그람작성법에서와 같이 C++는 프로그람작성자가 2 진수나 단항부호연산자를 리용하여 복잡한 식을 쓰게 한다. 실례

```
int i = 4 ;
int j =5 ;
i + 2 * j ;
```

에서 코드부분을 해석하여 보자. 연산순서에 따라 여러개의 가능한 결과가 있다.

왼쪽으로부터 오른쪽으로 가는 순서로 연산자를 적용하면 결과는 <30, int>이다. 오른쪽으로부터 왼쪽으로 가는 순서로 연산자를 쓰면 결과는 <14, int>이다. 이것은 명백히 연산자를 쓰는 순서에 대한 규칙들이다. 이 규칙들을 언어의 련관성 및 우선권규칙이라고 한다.

우선권을 보기로 하자. 매 연산자는 우선권준위를 가지고 있다. 표 2-8 에 흔히 론의하는 옹근수형 산수연산자의 우선권준위를 주었다. 일반적으로 더 낮은 우선권을 가진 연산자전에 더 높은 우선권을 가진 연산자를 쓴다. 본질적으로 산수식은 수학에서와 같다. 두 단항부호연산자는 가장 높은 우선권을 가지며 곱하기, 나누기, 나머지는 더하기와 덜기보다 더 높은 우선권을 가진다. 식

```
i +2*j ;
```

에서 곱하기는 더하기보다 더 높은 우선권을 가지므로 곱하기는 첫번째이고 연산결과는 값 14 를 만들기 위하여 i 의 값에 더해 진다.

표 2-8. 연산자우선권과 련관성

연산자	연산	우선권	련관성
+ -	단항더하기와 단항덜기	15	오른쪽
* / %	곱하기, 나누기, 나머지	13	왼쪽
+ -	더하기와 덜기	12	왼쪽

다른 실례로서 다음의 식을 생각해 보시오.

```
2/3+5 ;
-8*4 ;
8+7%2 ;
```

첫식에서 나누기는 더하기보다 더 높은 우선권을 가지므로 나누기를 먼저 수행한다. 결과는 0 인데 5 를 더하여 마지막값 5 를 얻는다. 두번째 실례에서 단항덜기연산자는 8 에 적용되며 -8 결과값에 4 를 곱하여 최종값 -32 를 얻는다. 세번째 식에서 나머지연산자는 수행되고 값 1 을 만든다. 식의 최종값은 9 이다.

흔히 C++우선권규칙는 무시해야 한다. 실례로 5 개 항목에 대한 판매세액을 계산하는 식을 쓰는것을 생각해 보시오.

식

```
Price1+Price2+Price3+Price4+Price5*0.06
```

은 명백히 정확치 않다. 곱하기연산자는 Price5 에 쓰고 결과는 다른 4 개 세액에 더한다.

C++는 식을 괄호안에 넣을것을 요구한다. 괄호로 둘러 막힌 식을 먼저 평가한다. 괄호를 사용하여 식을

```
(Price1+Price2+Price3+Price4+Price5)*0.06
```

과 같이 쓸수 있다.

이 식에서 5 개 세액을 먼저 합하고 그다음 합의 6%를 계산한다. 괄호로 막은 식은 겹칠수 있다. 다시 말하여 괄호로 막은 식은 다른 괄호로 막은 식을 포함할수 있다. 이가운데서 제일 안에 있는 괄호로 막은 식을 먼저 평가한다. 다음의 식을 생각해 보시오.

```
(2+(3+2)*5)/(4-2);
```

괄호로 막은 부분식 (3+2)는 다른 괄호로 막힌 식과 겹쳐 먼저 평가된다. 이때 괄호밖에 있는 식이 평가될수 있다. 최종값은 13 이다.

2.10.7 련관성

식 3*5/2 ; 을 보자. 이 식에서 연산자는 같은 우선권준위를 가진다. 곱하기를 먼저 또는 후에 수행하는가 하는데 따라 식의 값은 7 아니면 6 이다. 연산수가 우선권준위가 같은 연산자로 둘러 막힌 경우 연산수에 작용하는 연산자가 어느것인가를 아는것이 필요하다.이 특성을 연산자의 련관성이라고 한다. 그러므로 우의 실례에서 연산수 5 는 왼쪽연산자에 관계되며 식의 정확한 값은 7 이다(옹근수나누기는 자르기한것이라는것을 기억하시오). 다음의 코드부분을 보시오.

```
int t1 = 17 ;
int t2 = 3 ;
int t3 = 7
t1/t2 *5/t3 ;
t3*(-5/2)+t2 ;
t3+t1*4+t1 ;
```

이 식의 값들은 각각 1, -11, 92 이다.

괄호의 사용

우선권의 원리를 아는것은 중요하지만 필요하지 않는 부분까지 평가순서를 만들기 위해 괄호를 사용할 필요는 없다. 이것은 프로그람작성자에게는 어느 정도 도움이 될런지는 모르지만 후에 코드가 변화되면 다른 프로그람작성자들이 오유를 찾아 내는데는 그리 적합치 않다.

괄호 없는 식

```
a*b+c/d-3.0;
```

과 괄호 있는 식

```
(a*b)+(c/d)-3.0;
```

을 보자. 두식은 정확히 같은 계산을 수행하지만 두번째의 의미가 훨씬 더 명백한것은 사실이다. 이 책의 코드에서 복잡한 식의 명백한 평가순서를 만들기 위해서만 괄호를 사용한다.

문 제

20. 표시법 <value, type>를 리용하여 C++식 23.0+8 의 값을 구하시오.
21. 표시법 <value, type>를 리용하여 C++식 10/12 의 값을 구하시오.
22. 표시법 <value, type>를 리용하여 C++식 23%6 의 값을 구하시오.
23. 표시법 <value, type>를 리용하여 C++식 15/14 의 값을 구하시오.
24. 표시법 <value, type>를 리용하여 C++식 10L/2 의 값을 구하시오.
25. 표시법 <value, type>를 리용하여 C++식 25.5L/5 의 값을 구하시오.
26. 표시법 <value, type>를 리용하여 C++식 5/2 의 값을 구하시오.
27. cm 를 인치로 바꾸는 C++프로그람을 작성하시오. cm 에 대한 인치의 비률은 2.54 로 가정하시오.
28. <value, type>표시법으로 C++식 2+5L 의 값을 계산하시오.
29. <value, type>표시법으로 C++식 2.3f+5.2 의 값을 계산하시오.
30. <value, type>표시법으로 C++식 3.4L+3L 의 값을 계산하시오.
31. <value, type>표시법으로 C++식 2+5L 의 값을 계산하시오.
32. C++식 3/2+5 의 값을 구하시오.
33. C++식 5/1+2 의 값을 구하시오.
34. C++식 3*2+4*5 의 값을 구하시오.
35. C++식 4-2+5/3+2 의 값을 구하시오.
36. C++식 10-2+7%2-1 의 값을 구하시오.

2.11 출력명령

여기서는 간단한 옹근수와 류점수객체를 서술하고 초기화하여 이 객체들을 계산하는 방법을 본다. 이 기능은 계산결과를 현시할수 없다면 그리 쓸모 없다. 사용자에게는 프로그람에 의하여 계산된 정보를 출력하는 기구가 요구된다.

FORTRAN, PASCAL, BASIC 와 같은 다른 언어와 달리 C++는 출력을 위한 특수한 언어구조를 가지지 않는다. 오히려 출력능력은 표준 C++를 사용하여 수행하는 서고에 의하여 제공된다.

프로그람 2-1 에서 C++명령문

```
cout << "Hello ,world !" << end1 ;
```

은 화면에 통보문

Hello, world !

를 현시한다고 생각해 보자.

요구하는 효과를 만드는 명령문을 구체적으로 서술하는데는 여러가지 더 좋은 형식이 있을수 있으므로 여기서 모든 세부를 론의하지는 못한다. 그러나 명령은 C++의 풍부한 형태의 하나를 례증한다. 작업방법을 완전히 리해하지 않고도 다른것으로 개발한 객체를 사용할수 있다. 그러나 이 객체들을 호출하는 방법은 알아야 한다. 첫번째 프로그람은 인차 복귀하지만 더 간단한 실례를 보자. 다음의 명령을 보자.

```
cout << "C++";
```

우리가 호출하는 객체는 cout 이다. 그것은 출력명령객체이다. 명령이란 말은 장치로 가거나 장치로부터 나오는 자료흐름을 말한다. cout 는 출력명령객체이므로 장치로부터 자료를 읽지 않고 자료를 장치에 표시한다. 보통 cout 와 관계되는 장치는 현시장치이다.

cout 는 iostream 서고의 한 부분이다. 따라서 현시장치에 자료를 표시하기 위하여 명령으로 자료를 넣어야 한다. 이 지령은 추가연산자 <<이다. 우의 실례에서 문자 'C', '++'는 출력명령으로 추가된다. 그림 2-4 에 보여 주었다.

프로그람 2-1 에서 특수한 객체 endl 을 사용하였다. 출력명령으로 endl 을 쓰면 두가지 현상이 나타난다.

첫째로 행바꾸기문자('\n')가 지령으로 추가된다.

둘째는 명령으로 써준 모든 문자가 직접 현시장치에 옮겨 진다.

그러므로 출력은 출력장치를 지워 준다고 할수 있다.

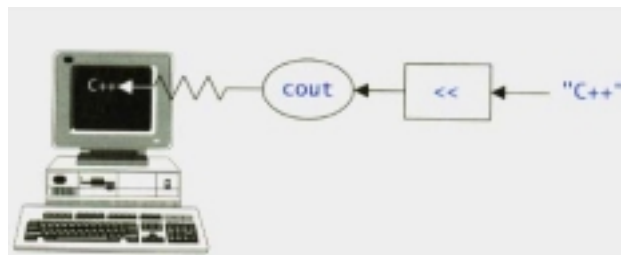



그림 2-4. cout 로 출력

endl 은 조종자라는 객체의 특수한 클라스이다.

조종자는 여러가지 효과를 가진 하나의 명령으로 추가할수 있는 객체이다. 우의 실례를

```
cout << "C++";
cout << endl ;
```

와 같이 쓸수 있으며 여기서 두번째 명령문은 문자렬 "C++"다음에 현시장치에 표시된다. 우의 삽입명령문들은 하나의 명령문으로 간단히 쓸수 있다. 산수연산자와 같이 추가연산자는 하나의 명령문에 출력값을 병렬로 줄수 있다. 우의 2 개 명령문을 다음과 같이 하나의 명령문으로 쓸수 있다.

```
cout<< "C++" << endl;
```

하나의 명령문에 종속할수 있는 추가연산자의 수는 제한이 없다. 다음의 명령문은 이 개념을 례증한다.

```
cout << "C++" << "is a" << "breeze" << endl ;
```

이 명령문은 현시장치에 통보

C++ is a breeze

를 출력한다.

cin 과 cout 의 결합

C++의 이전 판본들은 프로그람작성자들이 정수를 cin 으로 입력하기전에 cout 로 추가된 문자들을 명백히 없앨것을 요구하였다. C++의 현재판본은 cin 과 cout 결합관계가 없다. cout 명령은 정수연산을 만족시키기 위하여 cin 이 새 입력문자를 요구하는데로부터 정수연산때마다 자동적으로 지우기를 한다. 그리고 현시장치에 재촉문을 쓸 때 명백히 cout 로 지울 필요는 없다.

종속은 같은 명령문안에서 값이 서로 다른 행을 출력하여야 할 때 매우 쓸모 있다. 추가연산자는 임의의 C++기본형을 출력할수 있다. 실례로 명령문

```
cout << "18 % 4 =" << 18%4 << endl ;
```

은 다음의것을 출력한다.

```
18%4=2
```

명령문은 문자렬과 int 형값을 만드는 산수연산식의 결과를 출력한다. 우의 명령문은 또한 우선권에 대한 개념도 보여 준다. <<과 %연산자중 관계 있는 우선권준위에 따라 연산수 18 은 <<연산자 혹은 % 연산자로 둘러 막힌다는것을 명심할 필요가 있다.

다행히 %는 <<보다 더 높은 우선권을 가지므로 명령문은 자기가 하여야 할것을 한다. 그러나 우에서 지적한것처럼 명령문의 의미를 명백히 하기 위하여 식 18%4 를 괄호로 둘러 싸는것은 좋은것이다. 객체의 값을 출력하기 위하여 <<연산자도 사용할수 있다. 코드부분

```
int Hours = 11 ;
cout << Hours << "hours is" << (Hours *60) << "minutes" << endl ;
```

이 실행되면

```
11 hours is 660 minutes
```

가 출력된다. 이 코드에서 <<연산자는 명령 cout 에 int 객체연산자, 여러개의 문자렬연산자, 산수연산값의 결과출력을 조종한다. 출력연산자는 또한 류점수값을 출력할수 있다.

식

```
cout << (5.0/2.0) << " " << (1.0/3.0) << endl ;
```

은 현시장치에

```
2.5 0.33333
```

을 표시한다. 류점수값이 출력명령에 추가되면 출력연산자는 적은 화면공간에 값을 출력하게 한다. 따라서 코드부분

```
float x = 6.0 ;
cout << (x/2.0) << endl ;
```

은 현시장치에 3 을 쓴다. 다음에 류점수값의 표시를 조종하는 여러가지 기능을 고찰한다. 더 좋은 프로그람을 작성하기 위하여 iostream 서고를 리용하여 출력하는 다른 부분을 소개한다.

다음의 점들을 기억하여야 한다.

- 프로그람의 시작에 iostream과 string체계머리부파일을 포함하여야 한다.
- 이름공간 std내에서 객체를 리용하고 있다는것을 가리켜야 한다.
- <<연산자는 C++기본형을 출력할수 있다.
- <<연산자는 하나의 명령문에 여러개의 값을 출력하는데 종속시킬수 있다.

2.12 평균속도계산

마지막부분을 학습하기 위하여 도로표시판이 있는 도로에서 달리는 승용차의 평균속도를 계산하는 프로그람을 작성하자.

프로그람에 대한 입력은 리정표의 시작과 끝, 시간이다. 시간은 시간, 분, 초로 입력된다. 프로그람은 시간당 마일로서 평균속도를 계산하고 출력한다. 문제는 승용차의 평균속도를 시간당 마일로서 계산하라는것이다. 프로그람의 입출력동작은

```
All inputs are integers !
Start milepost ? 321
Elapsed time (hours minutes second) ? 2 15 36
End milepost ? 458
Car traveled 137 miles in 2 hrs 15 min 36 sec
Average velocity was 60.6195 mph
```

이다. 이 문제를 풀기 위한 단계는 다음과 같다.

단계 1. 재촉문을 나타내고 입력값을 읽는다.
단계 2. 지나간 시간을 계산한다.
단계 3. 이동한 거리를 계산한다.
단계 4. 평균속도를 계산한다.

매 단계를 일반적으로 변환할수 없으나 구체적인 조작까지 수행할수 있는 C++코드로는 쉽게 변환할수 있다. 단계 1을 수행하기 위하여 입력을 기억하는 객체의 형을 선택하여야 한다. 프로그람의 입출력동작은 모든 값이 옹근수로서 입력되는것을 보여 준다. 이렇게 시작점, 시작과 끝시간값을 기억하기 위하여 일반적으로 **int** 형을 사용한다. 이 방법을 사용하면 단계 4에서 생기는 중요한 문제에 대하여 알수 있다. 객체의 평균속도를 계산하는 공식은

$$속도 = \frac{거리}{지나간시간}$$

로 표시할수 있다. C++에서 나누기는 자르기를 포함한다. 대체로 자르기는 문제가 아니지만 지나간 거리보다 지나간 시간이 상대적으로 대단히 클 때 나누기는 0 결과를 가져 올것이다. 이러한 문제는 대부분 프로그람작성에서 제기되는 어려운 문제들중의 하나로 된다. 보통 어떤 부분을 조종하는 코드를 작성하기는 쉽지만 드문 경우를 예견하고 조종하는것은 어렵다.

있을수 있는 드문 경우를 예견하지 못하는것은 많은 프로그람에서 오유의 원인이다. 오유형태는 사용자의 입력과 호상 작용하는 프로그람에서 특수한 효과를 나타낸다.

경험

입출력과정에서 사용자와의 대화과정

대화하는 프로그람을 작성할 때 사용자에게 무엇을 하여야 하는가를 알려 주는것이 중요하다. 실례로 프로그람 2-10 에서 사용자는 반경을 입력하여야 하는데 그 값은 실수이다. 이 대기문은 필요없어 보이지만 프로그람을 작성하였으므로 실수를 입력하여도 원천코드는 정확히 동작한다. 그러나 어떤 사람은 원천을 보지 않고 프로그람을 실행하므로 사용자에게 프로그람의 요구를 알리는것은 중요하다. 프로그람 2-11 에 사용자의 요구를 주었다. 입력형태를 사용자에게 알려 주는것과 동시에 대화하는 입출력방법의 우점은 사용자에 의하여 입력이 정의된다는것이다.

이 련습은 사용자가 입력을 정확히 받고 판단하는것을 설명해 준다. 프로그람 2-10 에서는 면적에 따르는 반경을 계산하여 표시한다. 따라서 사용자는 프로그람이 정확한 입력을 받았는지 알수 있다.

이와 류사하게 프로그람 2-11 에서 출력값이 생겼을 때 거리와 지나간 시간은 계산된 속도에 따라 반영된다.

이 문제에 대한 2 개의 답이 있다. 모두 류점수객체를 만들수 있으며 따라서 모든 계산은 류점수산수연산으로 진행된다.

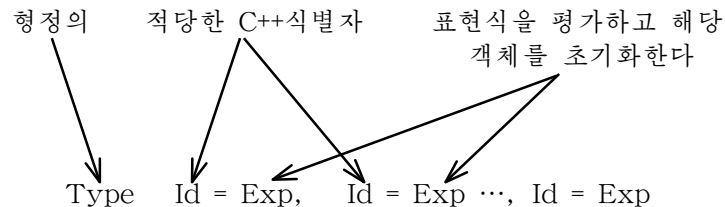
이러한 방법은 실지로는 가장 쉽지만 품이 더 든다.

더 좋은 방법은 류점수결과가 필요하지만 어려우므로 옹근수를 입력하고 류점수결과를 계산하게 하는것이다. 실례로 지나간 시간을 표시하는 하나의 값으로 시간, 분, 초를 입력하여 시간을 변환시켜야 한다. 변환은 정의로 할수 있다.

```
float ElapsedTime=EndHour+(EndMinute/ 60.0)+(EndSecond/3600.00);
```

나누는 수를 류점수형상수로 사용하였기때문에 나누기는 둘 다 류점수이며 더하기도 역시 류점수결과를 가져 온다.

우의 정의는 2.9 에서 보여 준 형태와 약간 차이난다. 거기서 초기값은 한 문자일수 있었다. 실지로 C++는 초기값에 아무런 식이나 다 사용한다.

따라서 일반형식은

이며 여기서 Exp 는 독자적인 식이다.

다음의 정의로 속도를 계산할수 있는데

```
float Velocity = Distance / ElapsedTime ;
```

이며 Distance 는 정의로부터 먼저 계산되였다.

```
int Distance = EndMilePost - StartMilePost ;
```

프로그람 2-11 에 완성된 프로그람을 주었다.

문 제

37. 명령문 cout 에서 식 25/4 의 값을 넣는 C++명령문은 무엇인가 ?
38. iostream 서고를 사용하는 C++프로그람안에 포함하여야 할 2 개의 파일은 무엇인가 ?

39. 대기문을 표시하고 시간을 받고 밤부터 지금까지의 초수를 계산하는 C++프로그람을 작성하시오. 시간은 HH:MM:SS 형식으로 입력한다.

```
// 프로그람 2-11: 승용차의 평균속도계산
#include <iostream>
#include <string>
using namespace std;
int main() {
    cout<< "All inputs are integers!\n" ;
    cout<< "Start milepost?" ;
    int startMilePost ;
    cin >> startMilePost ;
    cout << "End time (hours minutes second)?" ;
    int EndHour,EndMinute, EndSecond ;
    cin>> EndHour >> EndMinute>> EndSecond;
    cout<< "End milepost?";
    int EndMilePost;
    cin>> EndMilePost;
    float ElapsedTime=EndHour+(EndMinute/60.0)
      +(EndSecond/3600.0);
    int Distance=EndMilePost-StartMilePost;
    float Velocity=Distance/ElapsedTime;
    cout<< "\n Car traveled" << Distance << "miles in" ;
    cout<< EndHour << "hrs" << EndMinute << "min"
      << EndSecond << "sec\n" ;
    cout<< "Average velocity was" << Velocity << "mph"
      << endl;
    return 0;
}
```

프로그람 2-11. 승용차의 평균속도계산

알림

설명문의 배치

순서형식프로그람에서 약속은 함수의 시작에 모두 설명문을 두는것이였다. C++에서는 함수에 대체로 설명문을 쓴다. 함수의 형태에 따라 설명문을 주는데는 여러가지 방법이 있는데 각각 우단점이 있다. 객체의 첫 사용시점에 설명을 주는것은 객체형을 결정하기 위하여 함수의 시작부터 돌아 보지 않아도 된다는 우점을 가진다.

추가적으로 코드를 변경시키면 객체가 더이상 필요없으므로 불필요한 설명은 지우는것이 훨씬 더 좋다. 여러해가 지난후 낡은 형식의 코드를 만나면 설명문이 없는것들이 많다. 다른 한편 중심위치에 모든 설명을 주는것은 일부 우점을 가진다. 설명문을 어디서 보는것이 옳은가를 알게 된다. 통합개발환경과 열람도구의 출현으로 이 우점은 의의가 적어 졌다. 코드에서 설명문은 항상 먼저 시작하는곳 또는 그 가까이에 놓는것이 좋다.

콤퓨터의 력사

네피어의 계산틀과 계산자

계산에 일찌기 큰 공헌을 한 사람은 네피어(John Napier , Baron of Merchison)(1550~1657)였다.

1600 년대까지는 계산을 대체로 손으로 하였다. 특수하게 곱하기와 나누기가 수행되였는데 개별적인 계산을 많이 요구하여 어려웠다. 네피어는 로그의 원리를 발명하여 곱하기와 나누기를 간단한 더하기와 덜기 연산으로 전환시켰다.

제곱지수의 더하기와 덜기(즉 $x^4 \times x^3 = x^7$ 과 $x^7 \div x^4 = x^3$)가 이 원리에 따라 널리 퍼졌다.

네피어는 로그계산을 목적으로 《네피어의 계산틀》이라는 기구를 개발하였다. 기구는 본질에 있어서 움직이는 렬로 곱하기표를 맞추는것이였다. 이 계산틀은 렬이 뼈나 상아로 되여 있기때문에 네피어의 골격이라고도 하였다. 비록 원시적이지만 네피어의 계산틀은 1600 년대이전에 필수적인 계산도구로 되였다.

로그의 발전과 네피어의 계산틀은 여러가지 《다음세대》계산도구를 만들었다. 1600 년대에 만들어 져 1970 년대이후까지 여전히 사용한 한가지 도구가 계산자이다. 계산자는 본질상 로그의 물리적구조물이다. 계산자는 고정된 2 개 부분사이를 미끄러지는 한개 부분으로서 3 개의 목재 또는 금속부분으로 구성되여 있다(그림 2-5). 눈금은 로그에 맞추어 부분품에 새겼다. 수들은 더하기와 덜기로써 곱하고 나눌수 있다. 유표는 결과를 읽는데 도움을 준다. 계산자가 고도로 정밀화되여 매우 많은 눈금을 가지고 있으며 매우 복잡

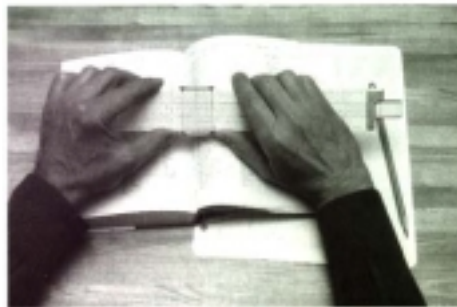



그림 2-5. 계산자

한 계산을 할수 있었다. 계산자에서의 한가지 문제는 정확성이 10 진수 4~5 자리로 제한되여 있는것이다.

2.13 알아 둘 점

- ✓ 명령문 #include <iostream>은 iostream 서고를 리용하여 프로그람이 입출력할수 있게 하는데 필요한 정의를 포함하는 직접적인 전처리이다.
- ✓ 명령문 **using namespace** std;는 사용할수 있는 이름공간 std 로 정의된 객체를 만든다. iostream 객체(즉 cin 과 cout)는 이름공간 std 에 정의되여 있다.
- ✓ C++연산자 <<는 추가연산자이다. 출력명령으로 본문을 추가할 때 리용한다.
- ✓ 출력명령 cout 는 보통 현시장치에 해당된다.
- ✓ C++연산자 >>는 추가연산자이다. 입력명령으로부터 문자들을 골라내기 위하여 사용한다.
- ✓ 입력명령 cin 은 보통 건반으로 실행된다.

✓ 조종자 endl 은 출력명령에 행바꾸기문자를 추가한다. 추가적으로 그것은 해당한 장치로 흐름을 보내는데 중심을 둔다.

✓ 표준 C++프로그람은 함수 main()에서 실행을 시작한다. 함수 main()은 프로그람이 성과적으로 실행되였는가 안되였는가를 지적하는 옹근수값을 돌려 준다.

✓ 값 0 은 성과적인 실행을 의미하고 1 은 프로그람의 실행시 문제나 오유가 생긴것을 의미한다.

✓ 값은 return 명령문을 리용하여 함수로부터 돌려 진다.

✓ 명령문 return 0 ;은 값 0 을 돌려 준다.

✓ C++설명문은 //로 시작하고 행의 끝까지 계속된다.

✓ 프로그람에서 효과적인 설명문을 쓰는것은 좋은 프로그람작성의 중요한 부분이다. 다른 프로그람작성자들이 프로그람을 읽을수 있다는것을 념두해 두어야 하기때문이다.

✓ 값주기연산자 =는 객체에 새값을 할당한다.

✓ C++객체형 short, int, long 은 옹근수형값을 준다. PC 에서 short 는 8bit 로 기억되고 int 는 16bit 로, long 은 32bit 로 기억된다.

✓ C++객체형 char 는 문자를 준다. 대부분의 장치들에서 문자들은 ASCII 문자모임을 리용하여 부호화된다. 부록 1 은 ASCII 문자모임을 포함한다.

✓ C++객체형 float, double, long double 은 실수값을 준다. PC 에서 long 은 32bit 에 기억되며 double 은 64bit 에 기억된다. 대부분의 PC 들에서는 long double 이 double 과 같은 크기이지만 다른 장치에서는 더 클수 있다. 실례로 일부 장치들에서 long double 은 128bit 이다.

✓ C++문자렬상수는 인용괄호로 둘러 막힌 문자렬이다. 행바꾸기, 타브, 경보와 같은 특수문자는 확장문자렬을 리용하여 문자렬에 포함할수 있다. 이것은 표 2-1 에 주었다.

✓ C++옹근수형상수는 3 가지 진수 즉 8 진수, 10 진수 혹은 16 진수중 하나로 쓰일수 있다. 8 진옹근수형상수는 o 으로 시작한다. 따라서 상수 o40 은 8 진수이며 10 진수 32 를 표시한다. 10 진수형상수는 0 이 아닌 다른 수자로 시작하며 16 진수는 앞붙이 0x 로 시작한다. 상수 0x40 은 10 진수 64 를 표현한다.

✓ C++는 류점수형상수를 쓰는 여러가지 방법을 지원한다. 가장 간단한 방법은 표준 10 진수표기법을 사용하는것이다. 3.1416, 2.53, 0.3512 류점수형상수는 또한 과학적인 표기법을 사용할수도 있다. C++는 류점수형상수 2.3E5 를 값 2.3×10^5 혹은 230000 으로 표시한다.

✓ C++이름은 문자(큰 글자와 작은 글자)렬, 수자, 밑선으로 구성된다. 정확히 이름은 수자로 시작할수 없다.

✓ C++이름은 정밀해야 한다. 실례로 이름 Temp 와 temp 는 2 개의 서로 다른 객체로 인식된다.

✓ 프로그람에서 객체의 이름을 의미 있고 서술적으로 주는것이 중요하다. 설명적인 이름은 다른 프로그람작성자가 프로그람의 동작을 리해하는데 도움을 준다.

✓ 객체는 사용되기전에 정의되여야 한다. 준비된 프로그람작성자들은 정의할 때 객체에 초기값을 준다.

✓ 옹근수나누기는 늘 소수점아래를 자른 결과를 만든다. 식 5/2 는 결과 2.5 가 아닌 2 를 만든다.

✓ 일반단항변환은 char 형 혹은 short 형연산수가 연산에 앞서 int 형으로 변환된다는것을 의미한다.

✓ 2 개의 옹근수형연산수를 포함하는 산수연산을 위하여 연산수가 서로 다른 형을 가질 때 덜 정확한 연산수를 더 정확한 연산수와 같은 형태로 만들기 위해 일반 2 진변환을 한후 결과를 연산해 낸다. 따라서 float 형과 double 형연산수들을 포함한 더하기연산에서 float 연산자는 double 로 변환되고 double 형더하기가 수행된다.

✓ 혼합산수식은 옹근수형과 류점수형연산수를 포함한다. 옹근수형연산수는 류점수형연산수로 변환되며 적당한 류점수형연산이 수행된다.

✓ C++의 우선권원리는 연산수에 작용하는 연산자의 순서를 정의한다. 산수연산자에 대하여 가장 높은 데로부터 가장 낮은 우선권은 단항 +와 -, *, /, 나머지, +와 -이다.

련습문제

2.1 32bit 옹근수의 범위는 얼마인가?

2.2 다음문자렬의 마지막에 null 바이트는 몇개인가?

"What's going on here ? \0"

2.3 두 int 형옹근수나누기가 어떻게 넘침결과를 만들수 있는가를 설명하시오.

2.4 프로그람 2-1 에서 명령문

```
return 0 ;
```

을 소거하시오. 수정된 프로그람을 콤파일하시오. 콤파일러는 오유나 경고를 알리는가? 그러면 통보는 무엇인가?

2.5 프로그람 2-1 에서 명령문

```
#include <iostream>
```

을 소거하시오. 수정된 프로그람을 콤파일하시오. 콤파일러는 오유나 경고를 알리는가? 그러면 통보는 무엇인가?

2.6 프로그람 2-1 에서 명령문

```
using namespace std ;
```

를 소거하시오. 수정된 프로그람을 콤파일하시오. 콤파일러는 오유나 경고를 알리는가? 그러면 통보는 무엇인가?

2.7 다음의 코드부분에 의하여 무엇이 표시되는가 ?

```
int i=2 ;
int j=3 ;
i =j + j ;
j = i * 1.5
cout << "i = " << i << "j =" << j << endl ;
```

2.8 프로그람 2-1 에서 include 명령문을 없애고 수정한 프로그람을 콤파일하시오. 콤파일러가 먼저 오유를 알리는것이 수정된 프로그람의 어느 행인가?

2.9 폰드로 객체의 질량을 받고 kg으로 객체의 질량을 출력하는 프로그람을 작성하시오.

2.10 객체의 체적을 계산하는 프로그람을 작성하시오. 프로그람은 객체의 질량과 밀도를 입력할것을 알린다. 질량은 g로 주며 밀도는 cm^3당 g로 한다. 질량, 밀도, 체적사이의 관계는 다음과 같다.

$$밀도 = \frac{질량}{체적}$$

프로그람은 cm^3로 체적을 출력한다.

2.11 알루미니움블로크의 질량을 계산하는 프로그람을 작성하시오. 프로그람은 cm로 블로크(즉 길이, 너비, 높이)의 치수를 입력한다. 알루미니움의 밀도는 $2.79g/cm^3$ 이다.

2.12 Price를 **int**형으로 하기 위하여 프로그람 2-2를 수정하시오. 프로그람을 실행시키시오. 프로그람 출력이 왜 달라 지는가를 설명하시오.

2.13 C++에서 확장문자렬이 아닌 문자에 빗선을 친 결과는 없다. 콤파일러들은 이 오유를 잡지 않는다. 콤파일러로 할수 있는껏 하여 결과를 알리시오.

2.14 모든 객체를 **float**형으로 사용하기 위하여 프로그람 2-11을 수정하시오. 프로그람의 입출력동작은 같은가?

2.15 다음의것들가운데서 알맞는 C++식별자는 어느것인가 ?

1) GPA	9) T2	17) A
2) Grade.put	10) 3CPO	18) _dog
3) GradePtAvg	11) Avg_Cost	19) Not !
4) int	12) $cost	20) _123
5) lstNum	13) Era	21) Cats
6) Num1	14) int	22) main
7) x-ray	15) PDQBach	23) Cost$
8) R2D2	16) ReturnV	

2.16 다음의 식의 결과는 무엇인가? <value, type>로 대답하시오.

1) 25/7	7) 30L%5
2) 21/3	8) 7-21
3) 26/2L	9) 28+3*5
4) 14%3	10) (27/3)+15
5) 31%3	11) 2
6) 22.1+1.0	12) -23+7*2

2.17 화씨온도를 받아 대응하는 섭씨온도를 출력하는 프로그람을 작성하시오. 섭씨온도에 대한 화씨온도의 변환비률은

$$°C = \frac{5}{9}(°F - 32)$$

2.18 섭씨온도를 받아서 그에 대응하는 화씨온도를 출력하는 프로그람을 작성하시오. 화씨온도에 대한 섭씨온도의 비률은

$$°C = \frac{5}{9}(°F - 32)$$

2.19 다음의 서술들을 가정하시오.

```
float f1 = 23.3 ;
float f2 = 1.0 ;
double d1 = 3.1 ;
int i1 = 5 ;
int i2 = 10 ;
int i3 = 7 ;
short s1 = 11 ;
short s2 = 5 ;
char c1 = '0';
```

이때 다음의 식의 결과는 무엇인가? <value, type>형식으로 대답하시오.

1) f1+d1	9) i2 + i3 +3.0
2) i1+d1	10) i2*f2+4
3) i1+i2*i3	11) s1 /i3
4) i2 % i3	12) c1+f2
5) i3/ i2 + i1*i3	13) s1+s2
6) f1-f3	14) i3+c1
7) f1-i3	15)c1-s2
8) f1/f2 +d1	

2.20 다음의 대수공식과 같은 C++식을 쓰시오.

1) b^2 +4ac	5) $-(a^2 - b^3)$
2) a +b/c +d	6) a(b/c)
3) $1/(1+x^2)$	7) (a+b)(c+d)(e+f)
4) $(4/3)\pi r^2$	

2.21 입력재촉문을 쓰고 km 로 거리를 읽고 mile 로 거리를 출력하는 프로그람을 작성하시오.

2.22 입력재촉문을 쓰고 5 개 옹근수를 읽고 평균을 계산하는 프로그람을 작성하시오.

2.23 다음과 같이 동작하는 C++명령문을 작성하시오.

$$Q = \frac{T1 \times T2}{D - K} + T2$$

값주기명령문은 다음의 객체들을 사용한다.

```
float q ;            // result
float k=1.25 ;       //constant of irritation
float D=9.2 ;        // duration
float T1=98.4 ;      //start time
float T2=101.12 ; //end time
```

2.24 입력재촉문을 쓰고 mile 로 거리를 읽어 km 로 거리를 출력하는 프로그람을 작성하시오.

2.25 입력재촉문을 쓰고 mile 로 객체의 이동거리와 그 거리를 가는데 걸린 시간을 읽는 프로그람을 작

성하시오. 프로그람은 객체의 속도를 계산하고 표시한다.

2.26 10 과 12 사이의 long 형옹근수를 받고 오른쪽으로부터 시작하여 매개 세번째 수마다 반점을 써주는 프로그람을 작성하시오.

2.27 입력재촉문을 쓰고 류점수를 읽어 다형성을 평가하는 프로그람을 작성하시오.

$3x^4 -10x^3 +13$

프로그람은 읽은 수와 다형성평가결과를 둘 다 표시한다.

2.28 입력재촉문을 쓰고 류점수를 읽고 다형성을 평가하는 프로그람을 작성하시오.

$23x^5 -6x^4 +11$

프로그람은 읽은 수와 다형성평가결과를 둘 다 표시한다.

2.29 입력재촉문을 쓰고 나이를 년도로 읽고 날자를 출력하는 프로그람을 작성하시오. 1 년은 365 일이라고 가정한다.

2.30 태양으로부터 지구에로 빛이 도달하는 시간을 계산하는 프로그람을 작성하시오. 빛의 속도와 지구와 태양사이의 거리를 알아야 한다.

2.31 사람의 나이와 심장박동수를 표시하는 프로그람을 작성하시오. 프로그람은 사람이 태여나서부터 심장박동수를 계산하고 표시한다. 1 년은 365 일이라고 가정한다.

2.32 모든 사람들이 한주일에 맥주를 2 병 마시게 한다. 맥주 한 상자(24 병)를 만드는데 보리가 12㎏이 요구된다. 1 천만명의 사람들에게 1 주일간에 필요한 맥주를 생산하는데 드는 보리의 량을 결정하는 프로그람을 작성하시오.

제 3 장. 객체들의 변경

소 개

이 장에서는 객체들과 프로그람작성자가 정의한 객체들을 수정하는 연산자들에 대하여 소개한다. 이러한 연산자에는 값주기연산자와 입력연산자가 있다. 부호는 값주기연산자이다. 값주기연산자는 대부분의 프로그람언어들에서 공통적이며 값주기에 대한 리해는 프로그람작성에서 반드시 준수해야 할 일반적인 개념으로 되고 있다. 입력연산자 >>는 흐름에서 값을 입력하여 객체를 설정하는 동작을 수행한다. 이 장에서는 프로그람작성자가 작성한 2 개의 객체형들인 SimpleWindow 와 RectangleShape 를 소개한다. 도형표시프로그람의 작성과정을 통하여 객체를 정의하는 여러가지 실례들을 볼수 있다.

기본개념

- 값주기연산자
- 값주기변환
- 값주기우선권과 결합
- 입력연산자
- 상수정의
- 합성값주기연산자
- cin 에 의한 입력
- 증가와 감소연산자
- 문자렬
- EzWindows

3.1 값주기

C++의 값주기연산자는 갈기부호이다. 이 연산자는 객체에 값을 설정한다.

```
int x = 10;
int y = 10;
```

이 코드는 x 라는 객체에 10 을 설정한다. 의미를 따지면 《x 가 10 이 된다》 혹은 《x 에 10 이 대입된다》라고 말할수 있다. 다음의 실례를 보면 더 잘 알수 있다.

```
float GrossSalary = 0.0;
float WithHolding = 0.0;
float TakeHomePay = 0.0;
GrossSalary = 50000.0;
WithHolding = GrossSalary * .05;
TakeHomePay = GrossSalary - WithHolding
```

이 코드에서는 3 개의 **float** 형의 객체를 정의하고 있다. 매 객체에는 기억기가 할당된다. 2 장에서 서술한바와 같이 C++번역기는 객체들에 기억기를 할당한다. 이러한 기억기들은 마치 《우편통》들로 생각할수 있다. 객체의 이름은 주소처럼 생각할수 있다. 그림 3-1 에서 왼쪽도표는 웃코드에서 값주기를 하기전의 기억기상태이다. 초기에 객체의 《우편통》에 0 이 들어 있다. 이 초기값들은 객체들을 정의할

때 설정된다. 그림 3-1 의 오른쪽도표는 값주기처리가 진행된후 기억기상태이다. TakeHomePay 의 값은 앞의 명령에서 계산된 WithHolding 의 값을 리용하여 계산되였다.

일반적으로 프로그람작성에서는 2 개 객체들의 값을 교환하여야 할 경우가 많이 제기된다. 실례로 아래와 같이 객체들을 정의하고 초기화한 다음 값을 교환하는 코드를 작성하여 보자.

```
int score1 = 90;
int score2 = 75;
```

그러자면 score1 의 값을 score2 에 복사하고 score2 의 값을 score1 에 복사하여야 한다.

```
score2 = score1;
score1 = score2;
```

만일 우와 같이 하면 score1 과 score2 의 값은 둘 다 score1 의 초기값 즉 90 으로 될것이다. 문제는 첫번째 값주기명령문에서 score2 의 값이 score1 의 값으로 설정되였기때문이다.

이런 현상을 막자면 객체의 값을 변경시키기전에 그 객체의 값을 림시 기억할수 있는 객체를 리용하여야 한다. Temp 라는 int 형객체를 정의하여 우의 실례를 해결하는 코드는 아래와 같다.

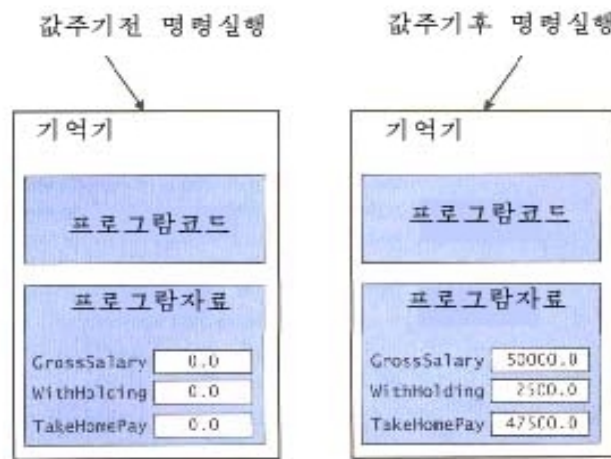
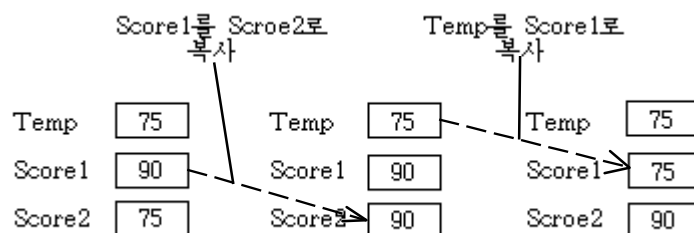
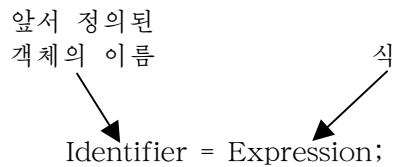



그림 3-1. 객체의 값을 수정한 값주기명령문해석

```
int Temp = score2;
score2 = score1;
score1 = Temp;
```

다음의 도표는 단계별처리공정이다.

3.1.1 값주기변환

앞절에서 가장 단순한 값주기명령문의 형태를 보았다.

원칙적으로 값주기연산자의 왼쪽, 오른쪽연산수는 자료형이 같아야 한다.

```
int x;
float y;
double z;
x = 1;
y = 3.2F;
z = 0.81;
```

아래코드와 같이 값주기의 왼쪽과 오른쪽연산수들의 자료형이 다르면 연산수들의 형을 바꾸어 같아지게 해야 한다.

```
int x = 0;
x = 2.3;
```

코드에서는 왼쪽연산수와 형이 일치하도록 오른쪽연산수를 변환하여야 한다. 이러한 변환을 값주기변환식이라고 한다. 실례로 아래의 코드에서

```
int x = 0;
int x = 2.3;
```

일 때 왼쪽연산수는 int 형이고 상수인 오른쪽연산수는 double 형이다. 값주기를 하기전에 값주기변환식의 전처리는 오른쪽연산수를 int 형으로 바꾼다.

류점수형을 옹근수형으로 변환하면 소수점아래부분이 잘리운다. 값주기변환식들은 요구대로 정확히 처리된다. 예견치 않았던 결과가 생기는것은 왼쪽연산수와 오른쪽연산수의 형이 다르면서 오른쪽연산수값이 왼쪽연산수의 기억능력보다 더 크기때문이다. 다음의 코드가 바로 그러하다.

```
short s1 = 0;
long i2 = 65536;
s1 = i2;
cout << "i2 is " << i2 << end1;
cout << "s1 is " << s1 << end1;
```

여기서 short 형은 16bit 이고 long 형이 32bit 이므로 출력결과는 다음과 같다.

```
i2 is 65536
s1 is 0
```

이 경우에 값주기변환에 의하여 높은 자리 16bit 는 무시되고 낮은 자리 16bit 만이 i2 에 값주기된다.

앞의 값주기연산자에서는 기억기에 값을 기억하는 한편 값주기연산자가 산수연산자와 같이 값도 만

든다는것을 강조하기 위하여 값주기표현식을 리용하였다.

x 는 int 형으로 선언하고 명령문에서 객체 x 에 1 을 주면 <1, int>결과가 얻어 진다.

```
x = 1;
```

값주기연산자의 결과형은 왼쪽연산수형이며 결과값은 왼쪽연산수에 기억된 값이다.

3.1.2 값주기우선권과 결합

값주기는 연산자이므로 x = y = z + 2; 과 같이 식을 쓸수 있다. 이 식은 무엇을 의미하는가? 그것은 값주기연산자의 우선권과 결합관계를 알려 주는데 있다.

먼저 우선권을 고찰해 보자. z 는 2 개의 서로 다른 연산자로 둘러 막혀 있기때문에 연산자의 상대적인 우선권에 따라 z 가 +에 결합되는가 아니면 =와 결합되는가를 결정한다. 일반적으로 z 와 2 를 더한다고 판단할것이다. 따라서 =의 우선권은 +의 우선권보다 낮다. 실지 =연산자는 제일 낮은 우선권준위를 가진다. 그러므로 웃식을 분석한다면

```
x = y = z + 2;
```

이다. 부록 1.2 에 모든 C++연산자의 우선권준위표를 주었다.

한편 우의 실례에서 y 는 =연산자로 막혀 있다. 이런 경우 y 가 왼쪽 =와 결합되는가 아니면 오른쪽 =와 결합되는가를 결정하여야 한다. 산수연산자와 달리 값주기연산자는 오른쪽결합이다. 따라서 웃식을 분석하면 x = (y = (z+2))이며 결과는 z+2 를 y 에 준것이 x 에 할당된다. 다시 말하여 값주기가 왼쪽결합이라면 일반적으로

```
(x = y)= (z + 2);
```

로 분석할수 있다.

알림

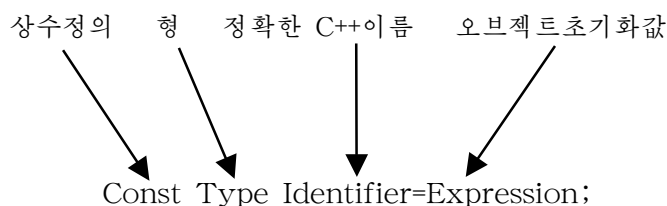
간단한 값주기형식

대체로 여러 객체들을 같은 값으로 초기화하여야 할 경우가 많다. 그때 객체들을 0 으로 초기화하는것이 대부분이다. 3 개의 int 형객체인 i, j, K 를 0 으로 초기화하는 코드는 아래와 같다.

```
i = 0;
j = 0;
k = 0;
```

우의 코드를 i = j = k = 0 으로 써도 된다. 이것이 더 간단할것이다. 그러나 그 효과는 달라지지 않는다.

3.2 상수정의

상수를 정의하면 아주 편리할것이다. 상수형객체의 정의형식은 다음과 같다.

실레로 다음과 같이 상수정의를 할수 있다.

```
const double AvogadroNmber = 6.02E23;
const float SpeedOfLight = 186000;
```

상수정의는 const 예약어를 제외하면 객체를 정의하고 초기값을 주는 일반객체의 정의와 같다. 예약어 const 는 객체가 초기값으로 설정된후에 다시는 수정될수 없다는것을 지적한다. 이것은 아주 효과적인 기능이다. 대부분의 프로그람들에서는 변경되지 않는 값에 일정한 문자들을 대응시킨다.

실레로 많은 과학자들과 프로그람기사들은 물리적인 상수(즉 아보가드로수, 빛속도)들을 리용하게 된다. 문자값을 리용하는것보다 값을 취한 const 객체를 리용하고 문자값보다 차라리 프로그람에서 객체의 이름을 리용하는것이 더 좋다. 실레로 프로그람 3-1 은 const 정의를 리용하여 프로그람 2-10 을 다시 쓴것이다. 객체 pi 는 값이 3.141519 인 상수이다.

```
// 프로그람 3.1:
#include <iostream>
#include <string>
using namespace std ;
int main() {
    cout << "Circle radius (real number)? *0"
    float Radius;
    cin >> Radius;
    const float pi = 3.14159;
    cout << "Area of circle with radius•0<< Radius
      << "is•0 << (Pi * Radius * Radius) << end1;
    cout << "Circumference is •0<< Pi * 2 * Radius << end1;
    return 0;
}
```

프로그람 3-1. 원의 면적과 둘레를 계산하는 프로그람

프로그람상수를 const 로 정의하는것은 아주 효과적이다. 실레로 상수를 수정하려고 하는 경우 해당처리부분만을 수정하면 된다. 프로그람 3-1 에서 π의 정확도를 더 높이자면 const 선언부분만을 바꾸고 다시 번역하면 된다. 그러나 프로그람 2-10 에서는 두곳의 내용을 변경시켜야 한다. 하나의 상수가 여러번 리용된 프로그람에서 모든것을 다 변화시킨다는것은 어려운 일이며 오유가 생길수 있다. 대신 상수의 이름을 리용한다면 프로그람을 리해하기가 보다 쉬워 질것이다.

경험

상수의 리용

프로그람실행시 변하지 않는 값인 const 객체를 리용하는것은 아주 좋은 습관이다. const 정의는 프로그람리용자에게 이 객체의 값이 변하지 않는다는것을 보여 준다. 또한 프로그람을 변경할 때 우연히 그 상수객체의 값을 변화시키는 명령문을 실행시키면 오유통보를 내보낸다. 또한 객체가 프로그람실행시에 변하지 않는 경우 언어처리기는 대체로 객체가 변경되는 경우보다 더 효률적인 코드를 발생시킬수 있다.

3.3 입력명령

2 장에서는 cout 객체와 출력연산자 <<를 리용하여 출력하는 방법을 서술하였다. iostream 서고는 입력을 위한 객체를 제공한다. 이 객체 cin 은 입력흐름객체이다. 그것은 대체로 건반과 관계된다.

즉 건반으로 입력한 기호들의 렬이 입력흐름을 형성한다. 입력흐름연산자 >>는 흐름으로부터 자료를 입력한다.

```
int value;
cin >> value;
```

코드는 cin 입력흐름에서 옹근수를 읽어 int 형객체 value 에 값을 넣는다. 이때 입력을 상세히 보기 위하여 여러가지 형변환처리를 할수 있다. 그러나 기본형들에 대한 입력연산자의 동작을 바로 리해하여야 한다. 옹근수들이나 류점수들을 입력할 때 입력연산자는 공백문자들을 뛰여 넘는다.

확장문자렬들로는 공백, 수직, 수평타브, 페지넘기기, 행바꾸기이다.

옹근수는 수자들 혹은 부호문자들(+, -)로부터 시작된다. 또한 류점수에는 소수점이 포함되여 있다. 암시적으로 입력연산자는 공백건으로 뛰여넘기하여 char 객체에 다음문자를 기억시킨다. 옹근수를 입력하는 경우 옹근수가 비확장문자로 시작하지 않는다면 어떤 현상이 일어 나겠는가? 이러한 경우에 입력흐름객체 cin 은 오유상태로 된다. 흐름객체가 오유상태일 때 흐름으로부터 추가한 문자들을 얻을수 없다. 다음의 입력은 수정된 객체의 값을 변경하지 못한다.

마지막장에서 입력처리에서의 오유검사방법과 회복방도를 론의한다. cin 의 처리를 설명하기 위하여 그림 3-2 와 같이 입력할 때 다음의 코드를 분석해 보자.

```
int ivalue;
cin >> ivalue;
float fvalue;
cin >> fvalue;
char cvalue;
cin >> c;
```

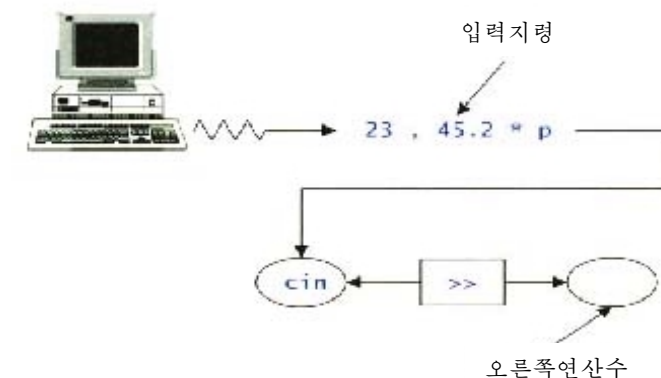



그림 3-2. 흐름 cin 의 입력과정

우선 입력연산자는 공백들을 뛰여 넘는다. 문자 "2"와 맞다들면 옹근수로 된 문자들이 읽어 져 옹근수형값으로 변환되며 이 값은 객체에 기억된다. 더이상 입력량은 랑비되지 않게 되며 이 경우 입력연산자들은 입력처리를 시작한다. 따라서 이때 객체 ivalue 는 23 으로 된다. 다음 연산자는 입력흐름으로부

터 류점수를 입력한다.

실례에서 공백이 아닌 다음문자는 반점이다. 이때 cin 은 오유상태로 되며 남은 입력연산자들은 아무런 효과도 없게 된다. fvalue 와 cvalue 는 둘 다 여전히 초기화되지 않은 상태로 남아 있다. 여러가지 형태로 될수 있는 옹근수를 생각하시오.

실례로 0x40 은 10 진수로 64 이다. 또한 8 진수로는 0100 이다. 입력연산자는 흐름에서 충돌하게 되는 이 수를 정확히 해석하기 위한 방법을 식별한다. 입력

```
0x52 034
```

는 Ivalue1 와 Ivalue2 객체가 다음의 코드토막으로 처리되면 10 진수값 82 와 28 을 주게 된다.

```
int Ivalue1;
int Ivlaue2;
cin >> Ivalue1 >> Ivalue2;
```

입력연산자는 반드시 출력연산자와 같이 될수 있다. iostream 서고는 입력자료에 대한 여러가지 기교를 제공한다. 실례로 가끔 입력파일에서는 모든 문자를 처리하여야 하며 아무 곳으로 뛰여 넘을수 없다. 이처럼 다른데서와 같이 입력된 수들을 해석하기 위한 입력연산자가 있어야 한다. 5 장에서는 여러가지 지령서고들에 대하여 충분히 보여 준다.

문 제

1. 정의 <값, 형>을 리용하여 값주기연산자의 왼쪽에 있는 객체에 할당된 값과 형을 구하시오.

```
int k;
k = 2.4;
```

2. 정의 <값, 형>을 리용하여 값주기연산자의 왼쪽에 있는 객체에 할당된 값과 형을 구하시오.

```
int j;
j = 5.9;
```

3. 정의 <값, 형>을 리용하여 값주기연산자의 왼쪽에 있는 객체에 할당된 값과 형을 구하시오.

```
int t;
T = 2.3L ;
```

4. 정의 <값 ,형>을 리용하여 값주기연산자의 왼쪽에 있는 객체에 할당된 값과 형을 구하시오.

```
float x;
x = 3;
```

5. 상수정의를 리용하는 기본리유는 무엇인가?
6. 2.4, 4 가 입력될 때 다음의 코드에서 객체 value1 과 value2 에 어떤 값이 들어 가는가?

```
int value1;
int value2;
cin >> value1 >> value2;
```

7. 7.8, 0 이 입력될 때 다음의 코드에서 객체 value1 과 value2 에 어떤 값이 들어 가는가?

```
float value1;
int value2;
cin >> value1 >> value2;
```

8. 23, 6이 입력될 때 다음의 코드에서 객체 value1과 value2에 어떤 값이 들어 가는가?

```
int value1;
int value2;
cin >> value1 >> value2;
```

3.4 탄화수소의 분자수계산

기초화학에서 일반적인 문제는 원자수나 분자수를 구체적으로 계산하는것이다. 여기서 흥미를 가지는것은 탄화수소이다. 이것은 2개의 원소 즉 탄소와 수소만을 포함하는 물질이다. 실례로 잘 알려 진 탄화수소는 메탄, 에탄, 프로판, 부탄 등이다. 탄화수소에 들어 있는 분자수계산에 대하여 례를 들어 설명하겠다.

임의의 물질 1몰에는 6.02×10^{23}개의 알갱이가 들어 있다는것을 상기하자. 이 수를 아보가드로의 상수라고 한다. 물질 1몰은 늘 g을 단위로 하여 질량으로도 표시된다. 몰질량은 원소들의 원자량의 합이다. 메탄 CH_4의 몰질량계산은 다음과 같이 한다.

탄소원자 1개	= 1 * 12.0 amu	=	12.0
수소원자 4개	= 4 * 1.0 amu	=	4.0
CH_4 1몰		=	16.0g

그런데 메탄 16g은 6.02×10^{23}개의 알갱이들을 포함한다. 다음의 같기식은 물질의 질량과 분자수사이관계를 나타낸다.

분자수 = 물질의 질량×몰질량/1몰× 6.02×10^{23}/1몰

적당한 량의 탄화수소에서 분자의 수를 계산하시오. 적당한 량의 탄화수소는 g으로 주어 진다. 탄소원자의 상대질량은 12amu이고 수소원자의 상대질량은 1amu이다. 프로그람의 입력과 출력에 대한 실례를 아래에 보여 준다.

```
Enter mass of hydrocarbon (in grams)
Followed by the number of carbon atoms
Followed by the number of hydrogen atoms
(e.g. 10.5 2 6): 16 1 4
is grams of a hydrocarbon
with 1 carbon atom(s) and 4 hydrogen atom(s)
contains 6.02e+23 molecules
```

이 알고리듬은 임의의 탄화수소의 분자수를 계산한다.

1단계. 입력재촉문을 쓰고 탄화수소질량과 탄소의 원자량 그리고 수소의 원자량을 읽는다.

2 단계. 1 몰의 질량(화학식량)을 계산한다.
3 단계. 선행한 공식을 리용하여 주어 진 질량의 탄화수소의 분자의 개수를 계산한다.
4 단계. 입력자료와 계산결과를 출력한다.

프로그람 3-2 는 이 알고리듬을 실현한 프로그람이다. 정의들과 출력, 입력을 진행한다.

```
cout << "Enter Hydrocarbon mass (in grams)\n"
"followed by the number of carbon atoms\n"
"followed by the number of hydrogen atoms\n"
"(e.g. 10.5 2 6):";
float Mass;
int CarbonAtoms;
int HydrogenAtoms;
cin >> Mass >> CarbonAtoms >> HydrogenAtoms;
```

알고리듬은 4 개의 단계를 수행한다. 질량은 옹근수값을 요구하지 않으므로 **float** 형으로 선언되였다. 입력연산자가 련달아 있다는데 주목을 돌리자. 이 기교는 3 개로 분리하여 쓰는것보다 더 효과적이고 더 짧다. 다음의 3 개 행은 수소 1 몰의 질량(화학식량)을 계산한다.

```
const int CarbonAMU = 12;
const int HydrogenAMU = 1;
long FormulaWght = (CarbonAtoms * CarbonAMU)
  + ( HydrogenAtoms * HydrogenAMU);
```

탄소와 수소의 원자량은 **const int** 로 정의된다. **const** 는 프로그람이 실현되는 기간 변경될수 없다는것을 언어처리기와 사용자들에게 알려 준다.

```
#include <iostream>
#include <string>
using namespace std;
int main ( ) {
    cout << "Enter mass of hydrocarbon (in grams)\n"
      << "followed by the number of carbon atoms\n"
      << "followed by the number of hydrogen atoms\n"·
      << "·(e.g. 10.5 2 6):"·;
    float Mass;
    int CarbonAtoms;
    int HydrogenAtoms;
    cin >> Mass >> CarbonAtoms >> HydrogenAtoms;
    const int CarbonAMU = 12;
    const int HydrogenAMU = 1;
    long FormulaWght = (CarbonAtoms * CarbonAMU)
      + ( HydrogenAtoms * HydrogenAMU);
```

```
    const double avogadroNumbr = 6.02e23;
    double Molecules = (Mass / FormulaWght)*AvogadroNumbr;
    cout << Mass << "grams of a hydrocarbon\n with"·
      << CarbonAtoms << "carbon atom(s) and"·
      << HydrogenAtoms << "hydrogen atom(s) \ncontains"·
      << Molecules << "molecules" << endl;
    return 0;
}
```

프로그람 3-2. 어떤 질량을 가진 탄화수소의 분자개수를 계산하기

탄화수소의 질량을 표시하자면 **int** 형으로는 부족하므로 **long** 형을 선택한다. 수소질량의 계산에서 결과

    (HydrogenAtoms * HydrogenAMU)

가 나타난다. 왜냐하면 HydrogenAMU 의 값이 1 이기때문에 우리는 이 식들이 필요없으며 전체 명령을

```
long FormulaWght = (CarbonAtoms * CarbonAMU)
  + HydrogenAtoms;
```

로 쓸수 있다는것을 알수 있다.

프로그람에서 중요한것은 다른 사람들이 그것을 쉽게 리해하고 수정할수 있도록 프로그람을 작성하는것이다.

우리는 가능한껏 정확한것을 요구하기때문에 아보가드로수나 분자수를 가지는 객체에 대하여 **double** 형을 사용한다.

알림

여러 행표시

프로그람 3-2 를 보면 화면이나 인쇄된 페지상에 출력된 코드를 보기가 무척 힘들다. 그것은 출력통보문의 길이가 매우 길기때문이다.실례로 아래의 코드를 보면

```
cout << "X -coordinate:" << Xcoord << "Y -coordinate:" << _
Ycoord <<"Z -coodinate:" << Zcoord << end1;
```

으로서 한개 명령문이 대단히 길다. 따라서 창문의 범위를 벗어 나게 된다.

다중행표현에서 긴 표현을 쓰지 않는 아래와 같은 좋은 성질은 그 연장선이 언제나 하나의 연산자에 의해 시작되며 그것이 하나의 빈 공백을 계약하게 된다는것이다. 이것들은 둘 다 앞행의 련속이라는것을 독자들에게 알려 주는데 리용된다. 레를 들면 앞의 실레

```
cout << "X -coordinate:" <<Xcoord
<< "Y-coordinate:" <<Ycoord
<< "Z-coordinate:" << Zcoord << end1;
```

를 다른 실레로

```
((Xcoord1 - Ycoord1) /2 * Distance1 + ((Xcoord2
  -Ycoord2) /2) * Distance2;
```

와 같은 긴 수학식들로 쓸수 있다.

수학식의 우점은

```
((Xcoord1 - Ycoord1) / 2) *Distance1
  + ((Xcoord2 - Ycoord2) / 2) *Distance2;
```

와 같이 될수 있다는것이다. 출력하는 동안 값이 출력되고 있는 나머지명령문들이 긴 문자렬이면 우리는 선택한 행들을 분할할수 있으며 다음행에서 그 문자렬이 계속된다.

언어처리프로그람은 하나의 큰 문자렬에서 함께 련결된 문자렬과 단정확도연산지령에서 출력하는것을 자동적으로 처리한다.

3.5 동시할당

C++에는 일반적인 연산들을 수행하기 위한 여러개의 특수한 연산자들이 있다. 이러한 연산자들은 성구나 언어의 속기법과 같이 생각할수 있다. 즉 하나의 객체에 대하여 2개의 연산자를 대응시킬수 있다.

실례로 i라는 객체에 5를 더한 다음 그 결과를 다시 i에 기억시킨다고 가정해 보자. 많은 프로그람언어에서는 이 연산을 i = i + 5라는 명령문으로 실현할것이다.

C++에는 이 동작을 수행하는 복합명령연산자가 있다. 즉 i+=5;라고 더 간단히 쓸수 있다. C++는 모든 2진산수연산자들에 대한 복합명령연산자들을 가지고 있다. 프로그람 3-3은 연산자의 이러한 형을 결정한다.

```
// 프로그람 3-3. 복합값주기실례
#include <iostream>
#include <string>
using namespace std;
int main() {
    int  = 5;
    int k = 2;
    i /= k;
    cout << "i is" << i << end1;
    int j = 20;
    j *= k;
    cout << " j is" << j << end1;
    int m = 15;
    m %= 4;
    cout << "m is" << m << end1;
    return 0;
}
```

프로그람 3-3. 복합값주기실례

이 프로그람의 출력은

```
i is 2
j is 40
m is 3
```

이다. 복합명령연산자의 같은 형태가 아닐 때 무엇이 나타나는가?

례를 들어 아래와 같은 코드부분을 분석하자.

```
int i = 10;
float y = 3.2;
i += y;
i = i + y;
```

무슨 변환과 무슨 연산이 수행되며 결과값의 형은 무엇인가?

이 질문들에 대답하기 위하여 i=i+j;와 같은 명령연산들을 생각하는것이 편리하다. 2 진연산자와 값주기에 대한 변환규칙을 리용할수 있다.

먼저 보통 연산수들의 단항변환과 2 진변환이 수행되고 다음연산이 수행된다. 실례에서 i 는 float 로 변환되면 <13.2, float>의 결과로 되며 float 형의 추가가 진행된다. 그다음 값주기할당이 끝난다. 따라서 13 이 i 에 기억된다. 간단한 할당과 같이 결과값은 <13.2, int>이다.

경험

속단하지 마시오.

대부분의 프로그람작성자들은 자기가 만든 프로그람을 락관하며 더 빨리 실현하려고 한다. 능력이 중요하다면 명확성과 정확성은 더 중요하다. 누가 틀린 결과가 없이 빨리 프로그람을 실행시키는가? 명확성과 정확성이 없이는 능력을 발휘할수 없다.

프로그람에서 첫째 원칙은 프로그람실행시간은 90%를, 코드작성시간은 10%를 차지한다는것이다. 따라서 대부분 시간을 보내는 프로그람에 대하여 아무 생각이 없이 프로그람을 실행, 변화시키는것은 전체 실행시간에 영향이 적거나 없다.

만일 능력이 기본으로 된다면 더 좋은 효과적인 방법은 프로그람이 완성될 때까지 기다리는것이며 그다음 프로그람의 결함을 판단하기 위한 프로파일러(profiler)라는 특수한 도구를 사용한다. 큰 결함은 대부분 프로그람실행시간에 있다. 프로그람들의 이러한 측면들은 프로그람실행시간을 바꾸는것에 귀착된다.

3.6 증가와 감소

C++도 역시 증가 또는 감소하는 하나의 객체에 대하여 특별한 연산자들을 가진다. 연산자 ++는 증가연산자, 연산자 --는 감소연산자이다.

C++라는 이름에도 까닭이 있다. 실지로 《증가된 C》이다. 수학적인 객체들이 적용되는데 이러한 연산자들은 객체의 함수로부터 하나를 덜거나 더한다.

실례로 아래의 코드부분

```
int i = 4;
++i;
cout << "i is" << i << end1;
```

의 출력결과는

```
i is 5
```

이다. 모든 목적과 지향에 대하여 값

```
++i
```

는

```
i+=1;
```

과 같지만 그것은 훨씬 더 짧다. 흥미 있게도 앞붙이와 뒤붙이의 증가와 감소연산자라는 2 개의 형태가 존재한다. 앞붙이의 형식은 연산자가 연산수의 앞에 놓인다는것이다. ++i 가 한가지 실례이다. 뒤붙이형태도 있다. 이전의 코드토막은

```
int i = 4;
i ++;
cout << "i is" << i << end1;
```

인데 이 토막과 똑같은 결과를 만들어 낸다.

증가연산자의 앞붙이와 뒤붙이사이의 차이는 무엇인가? 그 차이는 연산자가 하나의 큰 부분식으로 사용될 때 나타나게 된다. 이 코드토막을 고려하면

```
int i = 4;
int j = 5;
int k = j * ++i
cout << "k is" << k << ", i is " << i << end1;
```

이다. 곱하기연산은 i 가 증가하기전에 또는 후에 수행되는가?

++가 앞붙이연산자인 경우 i 가 먼저 증가하고 그다음 곱하기를 진행한다. 따라서 결과는 k is 25, i is 5 이다.

토막을

```
int i = 4;
int j = 5;
int k = j * i++;
cout << "k is" <<k ", i is" << i << end1;
```

으로 수정하면 출력은 k is 20, i is 5 로 될것이다. 즉 뒤붙이증가는 그것이 증가되기전에 객체의 값을 돌려 준다. 그런데 앞붙이증가는 그것이 증가한후에 객체의 값을 돌려 준다. 감소연산자는 하나를 던다는것을 내놓고는 증가연산과 비슷하다.

증가와 감소연산자는 류동소수점객체들인 float, double 과 long double 연산자들에 대하여서도 같은 식으로 적용된다. 그것들은 각각 그 객체로부터 하나를 덜거나 더한다. 코드토막을 보면

```
float f = 5.2F;
double d = 8.6;
++f;
--d;
cout << "f is" << f << ", dis" << d << end1;
```

이고 출력은 f is 6.2, d is 7.6 일것이다.

주목하여야 할 하나의 중요한 점은 증가와 감소연산자가 오직 객체에만 적용될수 있다는것이다. 실례로 그것은 표현 (x-2)++를 해석하면

```
x-2+1;
```

을 의미한다.

어쨌든지간에 이 식은 규정에 어긋나며 번역되지 않을것이다. 증가연산자는 하나의 표현에 적용되고 있다. 부록 1.2에서 증가와 감소연산자의 형태와 결합을 보시오.

알림

증가와 감소

주목하였던것처럼 증가와 감소연산자는 하나의 수학적객체로부터 하나를 더하거나 덜기 위한 C++속기법이다. 세련된 C++프로그람작성자는 i=i+1 과 같이 절대로 쓰지 않는다. 이러한 경우 앞붙이증가를 쓰는가 뒤붙이증가를 쓰는가를 볼 필요가 없다. 경험은 대부분의 세련된 C++프로그람작성자들이 앞붙이를 잘 쓴다고 볼수 있다.

문 제

9. <값, 형태>의 표기법을 사용하여 다음의 행의 왼쪽에 있는 객체에 할당된 값과 형태를 구하시오.

```
int i = 3;
float f = 6.1;
i += f;
```

10. <값, 형태>의 표기법을 사용하여 다음의 행의 왼쪽에 있는 객체에 할당된 값과 형태를 구하시오.

```
int i = 4;
float f = 6.8;
f += i;
```

11. <값, 형태>의 표기법을 사용하여 다음의 행의 왼쪽에 있는 객체에 할당된 값과 형태를 구하시오.

```
short i = 4;
int k = 6;
i -= k;
```

12. <값, 형태>의 표기법을 사용하여 다음의 행의 왼쪽에 있는 객체에 할당된 값과 형태를 구하시오.

```
int i = 5;
int j = 0;
int k;
k = ++i ;
j = i;
```

13. <값, 형태>의 표기법을 사용하여 다음의 행의 왼쪽에 있는 객체에 할당된 값과 형태를 구하시오

```
int i = 6;
int j;
int k;
k = i ++;
j = i;
```

14. <값, 형태>의 표기법을 사용하여 다음의 행의 왼쪽에 있는 객체에 할당된 값과 형태를 구하시오.

```
float x = 3.2;
float y;
float z;
```

```
y = x++;
z = x;
```

3.7 매해저금량의 평가

누구든지 돈의 여유가 생기면 그 돈으로 무엇인가 사려고 할것이다.

이 장에서는 4 주동안에 저금한 자료에 기초하여 년간에 저금할수 있는 돈을 평가하는 프로그람구조를 취급한다. 그 문제를 보자.

매주마다 얼마만큼 저금하였는가에 따라 해마다 저금한것을 계산하시오. 매주의 말에 저장한 일정한 량의 변화는 펜스(penny), 니클(nickel), 다임(dime) 쿠워터(quarter)라는 4 개의 화페단위로 기록된다. 여기에 입력과 출력의 프로그람동작실례가 있다.

```
For each week enter 4 numbers:
Pennies nickels dimes quarters(e.g.: 3 2 4 1)
Week 1 data: 8 2 5 3
Week 2 data: 4 3 3 5
Week 3 data: 8 5 6 3
Week 4 data: 5 2 7 6
Over four weeks you have collected
  25 pennies
  12 Nickels
  21 Dimes
  17 Quarters
                 Which is 7 dollar(s) and 20 cent(s)
This is a weekly average of 1 dollar(s) and 80 cent(s).
Estimated yearly savings: 93 dollar(s) and 60 cent(s).
```

이 문제를 풀기 위한 알고리듬은 명확하다.

단계 1. 입력재촉기호를 표시하고 매주자료를 읽고 저장된 펜스, 니클, 다임, 쿠워터의 총수를 계산하고 보관한다.

단계 2. 저장된 니클의 수를 인쇄한다.

단계 3. 저장된 총량과 주별평균을 계산하고 인쇄한다.

단계 4. 년말에 평가된 보관량을 계산하고 인쇄한다.

목록 3-1 에서는 알고리듬의 첫 두 단계를 수행하는 코드를 보여 준다. 프로그람은 사용자가 자료를 어떻게 입력하겠는가를 보여 주는것으로부터 시작된다. 다음의 4 개 부분들은 매주의 자료를 읽고 펜스의 총값을 계산한다. 총값을 갱신하는데 +=연산자를 사용한다는것을 기억하시오. 15 번째 부분은 총값을 인쇄한다.

목록 3-1. change.cpp의 한 부분

```
// 매해저금량의 평가
#include <iostream>
#include <string>
using namespace std;
int main( ) {
    cout << "For each week enter 4 numbers:\n"
    pennies nickels dimes quarters (e.g.: 3 2 4 1 )\n· << end1;
    cout << "Week 1 change:·0
    int Pennies, Nickels, Dimes, Quarters;
    cin >> Pennies >> Nickels >> Dimes >> Quarters;
    int TotalPennies = Pennies;
    int TotalNickels = Nickels;
    int TotalDimes  = Dimes;
    int TotalQuarters = Quarters;
    cout << "Week 2 change:·0
    cin >> Pennies >> Nickels >> Dimes >> Quarters;
    TotalPennies += Pennies;
    TotalNickels += Nickels;
    TotalDimes += Dimes;
    TotalQuarters += Quarters;
    cout << "Week 3 change:·0
    cin >> Pennies >> Nickels >> Dimes >> Quarters;
    TotalPennies += Pennies;
    TotalNickels += Nickels;
    TotalDimes += Dimes;
    TotalQuarters += Quarters;
    cout << "Week 4 change:·0
    cin >> Pennies >> Nickels >> Dimes >> Quarters;
    TotalPennies += Pennies;
    TotalNickels += Nickels;
    TotalDimes += Dimes;
    TotalQuarters += Quarters;
    cout << "Over four weeks you collected\n ·0<
      TotalPennies << "Pennies\n ·0<<TotalNickels
      << "Nickels\n "· << TotalDimes << "Dimes\n "
      << TotalQuarters << "Quarters" << end1;
}
```

프로그람의 2 번째 부분은 저장된 총량과 주평균(목록 3-2 를 보시오.)을 계산한다. 이 단계는 나누기와 나머지연산자를 사용하여 딸라의 수와 펜스의 수를 따로따로 계산한다. 그때는

```
int Totaldollars = Total / 100;
```

로서 딸라의 수를 계산한다.

```
int TotalCents = Total % 100;
```

은 펜스의 수를 계산한다. 이 연산자들의 렬은 한 딸라와 펜스의 량을 인쇄하여야 할 때 수행된다.

이 수법은 단계 4 에서 계산된 평균저금값과 년말에 보관한것을 평가하고 출력하는것이다.

목록 3-2. change.cpp 의 2 번째 부분

```
// 총 저장된것들과 주별평균을 계산하고 인쇄한다.
int Total = TotalPennies+TotalNickels*5+TotalDimes*10+TotalNickels*5
int Average = Total/4;
int TotalDollars=Total/100
int TotalCents = Total % 100;
int AverageDollars = Average / 100;
int AverageCents = Average % 100;
cout << "Which is " << TotalDollars << "dollar(s) and"
  << TotalCents <<"Cent(s) ." << end1;
cout << "This is a weekly average of"
  << AverageDollars << "dollars(s) and" << Averagecents
  << "Cent(s)." << end1;
// 년마다 평균값에 저장한것을 계산하고 인쇄한다.
int YearSavings = Average * 52;
int YearDollars = YearSavings / 100;
int YearCents = YearSavings % 100;
cout << "Estimated yearly savings:"
  <<YearDollars << "dollar(s) and"
  << YearCents << "cent(s)." << end1;
return 0;
}
```

3.8 string 클라스

지금까지 우리는 C++의 전통적인 객체들 **char**, **int long**, **float**, **double** 로써 작업하였다. 이 전통적인 객체들외에 C++언어의 정의에서는 하나의 표준서고를 더 지적한다. 이 서고는 많은 추가적인 쓸모있는 객체들을 포함하고 있다. 훌륭한 C++프로그람작성자가 되기 위하여서는 표준서고에서 어떤 편리한

것들을 사용할수 있으며 그것들을 어디에 쓰는것이 알맞는가를 배워야 한다.

하나의 문자렬은 하나의 단일한 객체로 만들어 진 문자들의 렬이다. 령문자렬이나 2 중괄호로 막힌 더 많은 문자들과 같은 문자렬상수들을 사용한다.

그런데 C++는 문자렬을 지정하고 저장하기 위한 기본객체를 제공하지 않는다. 다행스럽게도 표준 C++서고는 이 목적으로부터 한개의 클라스를 제공한다. 그 클라스 string 은 문자렬을 가질수 있는 객체로서 사용한다. string 클라스를 사용하기 위하여 프로그람은

```
#include <string>
```

와 같은 include 정의를 반드시 하여야 한다.

서고객체들은 기본객체와 함께 창조되고 정의된다. 실례를 들어 코드토막

```
string Greeting = "Hello";
```

은 Greeting 이라는 하나의 문자렬객체를 정의하고 인용괄호안에 있는 문자렬로서 그것을 초기화한다. 이 정의에서 주목하여야 할 점은 전통적인 또는 적당한 객체를 정의하고 초기화할 때와 같은 방법을 사용하여야 한다는것이다.

흥미 있는것은 하나의 문자렬을 하나의 문자렬상수로 초기화할수는 있지만 하나의 문자로는 초기화할수 없다는것이다. 아래의 정의는 틀린다.

```
string ExclamationMark = '!';
```

우리가 1 장에서 취급한것처럼 하나의 클라스는 그것이 리해하는 한조의 속성들과 통보문을 가진다. 클라스 string 은 아래의 속성들을 가진다.

- 문자렬을 이루는 문자들
- 문자렬의 문자수

결과적으로 우리는

```
string Message = "Help";
```

를 아래와 같이 서술된 객체를 창조하는것으로 정의할수 있다.

고정된 문자렬과는 달리 하나의 문자렬객체는 저장된 문자들을 끝내는 '\0'을 가지지 않는다.

string 객체가 해석할수 있는 동작이나 통보들과 string 객체들에서 수행할수 있는 여러개의 연산자들이 있다. 문자렬들은 출력과 입력연산자를 사용하여 쓰거나 읽을수 있다. 아래의 례는 Prompt 의 내용을 지령 cout 를 사용하여 쓰기한다.

```
string Prompt = "Enter your password";
cout << Prompt;
```

아래의 문자들이 화면에 표시된다.

Enter your password

아래의 코드토막은 지령 cin 으로 문자렬을 얻는다.

```
string Account;
string Password;
cin >> Account;
cin >> Password;
```

입력연산자는 초기공백을 지나서 string 연산수에 대하여 끝단어를 의미하는 공백을 읽는다. 그래서 우의 코드토막을 실행하고

Hot Stuff!

라고 쓰면 단어 Hot 는 Acount 에 기억되고 단어 Stuff!는 Password 에 기억된다. 전통적인 객체들과 같이 string 객체는 다음의 객체에 값주기할수 있다. 실례로 코드토막

```
string Message1 = "Hello!";
string Message2;
Message2 = Message1;
```

은 그림 3-3 에 보여 주는것처럼 Message1 의 내용을 Message2 에 복사한다.

결국 2 개의 서로 다른 문자렬에 같은 값이 존재한다.

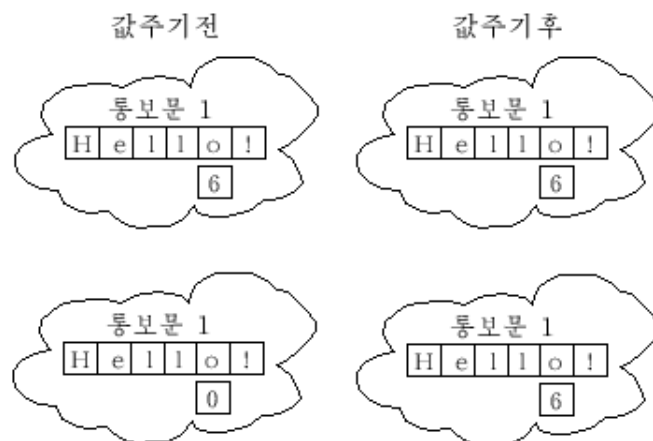



그림 3-3. 문자렬객체의 할당

추가적으로 string 클라스는 값주기연산자로 2 개의 문자렬을 련결하여 새로운 하나의 문자렬을 제공할수 있는 능력을 가진다. 련결은 하나의 문자렬뒤에 다른 문자렬을 붙이는 방법으로 2 개의 문자렬가운데서 하나의 새로운 문자렬을 구성한다. +는 문자렬련결연산자이다.

아래의 코드토막을 주의해 보시오.

```
string FirstName = "Zach";
string LastName = "Davidson";
string FullName = FirstName + ?? + LastName;
```

문자렬 FullName 은 Zach Davidson 으로 초기화된다.

련결연산자는 하나의 문자를 하나의 문자렬에 련결할수 있다.

클라스 string 에는 또 하나의 다른 연산자가 있다. 확장연산자 +=는 하나의 문자렬끝에 문자들을 추가한다. 련결과 같이 확장연산자는 또 하나의 문장이나 문자를 한 문자렬의 끝에 추가할수 있다.

코드토막을

```
string Message = "Help";
Message += '!';
cout << Message << end1;
```

로 하면

```
Help!
```

로 된다. 류사한 코드토막

```
string Message = "Help";
string who = "Me!";
Message += who
cout << Message << end1;
```

은

```
Help Me!
```

로서 출력한다.

string 객체는 연산자들에 대한 리해외에 많은 행동과 통보문들을 리해한다. 그중 대단히 많이 쓰이는것들만을 언급하자. 하나의 편리한 통보문은 string 객체가 해석하는 size 이다. 통보문은 문자렬객체가 문자렬에서 문자의 수를 돌려 줄것을 요구한다.

객체에 대응한 하나의 통보문을 보내는 C++문법은 다음과 같다.

C++토막

```
string Date = "March 7, 1994";
int i = Date.Size();
```

을 실행하면 i 에는 13 이 들어 간다. Size 통보는 아무런 인수도 요구하지 않는다. string 클라스에 의하여 제공되는 또 하나의 능력이 Substr 인데 문자렬가운데서 보조문자렬을 돌려 준다. 선택된 보조문자렬은 Substr 에 대한 설명에서와 같이 시작위치와 길이를 주어 정의한다. C++에서 문자렬의 첫 문자의 위치는 0 이다.

코드토막

```
string Date = "March 7, 1994";
```

```
string Year = Date.SubStr(9, 4);
cout << "Year is" << Year << end1;
```

에서는 Date 문자렬의 9 번째 문자로부터 시작하여 4 개 문자를 Year 에 대입한다. 즉 "Year is 1994"가 출력된다. 문자렬에 없는 위치를 Substr 에 주면 오유이다. 만일 지적된 위치와 요구된 부분문자렬길이의 합이 그 문자렬보다 크면 Substr 는 문자렬의 끝을 보조문자렬의 시작위치로 돌려 준다.

```
string Date = "March 7, 1994";
string Year = Date.Substr(9, 10);
cout << "Year is" << Year << end1;
```

즉 우의 코드토막은 다음과 같은 결과를 현시한다.

```
Year is 1994
```

클라스 string 은 또한 문자렬에서 하나의 부분문자렬을 탐색하는 기능도 가지고 있다. 이 통보문을 find 라고 한다. 이 find 통보문은 2 개의 변수 즉 검색문자렬과 검색시작위치를 받아 들인다. find 통보문은 발견된 문자렬의 위치를 돌려 준다. 만일 부분문자렬이 없으면 find 는 문자렬의 길이를 넘는 위치값을 되돌려 준다. 코드토막을 보면 다음과 같다.

```
string SpockSays = "Live long and prosper!";
int i = Spocksays.find("end", 0);
cout << "i is" << i << end1;
```

두번째 명령문은 문자렬 SpockSays 에서 부분문자렬을 탐색하여 0 을 되돌린다. 결과

```
i is 10
```

이라는 본문이 현시된다.

끝으로 string 서고는 하나의 문자렬이 입력된 모든 행자료를 읽게 하는 GetLine()을 제공한다. 현재 취급하고 있는 다른 string 의 기능과는 달리 GetLine()은 string 객체에 보내는 통보문이 아니라 string 서고에 제공된 보조적인 기능이다. 아래의 코드는 GetLine()을 리용하여 cin 으로부터 입력된 한개 행을 읽는 실례이다.

```
cout << "Please enter some text:";
string InputLine;
GetLine(cin, InputLine, '\n');
cout << "Your input is " << InputLine << end1;
```

GetLine()에서 첫 인수는 읽어 들이기 위한 흐름이고 두번째 인수는 입력행을 접수하기 위한 string 객체이며 세번째 인수는 읽어내기를 끝내는 문자기호이다. 우의 코드를 실행하였을 때 나타난 결과를 아래에 보여 주었다.

```
Please enter some text: a man a plan a canal panama
Your input is man a plan a canal panama
```

문자렬서고를 설명하기 위하여 아메리카형식날자(실례로 December 29,1953)를 국제형식날자(실례로 28 December 1953)로 변환하는 프로그람을 작성하자. 문제풀이는 간단하다. 아메리카형식으로 날자

를 입력하고 국제형식으로 날자를 출력하는것이다. 이 문제를 풀기 위한 알고리듬은 다음과 같다.

1단계. 지령대기문에 아메리카형식으로 날자를 입력한다.
2단계. 월을 얻어서 Month라고 하는 객체에 저장한다.
3단계. 날을 얻어서 Day라고 하는 객체에 저장한다.
4단계. 년을 얻어서 Year라고 하는 객체에 저장한다.
5단계. 날, 월, 년의 형태로 날자를 표시한다.

다음의 코드는 지령대기문상태에서 날자를 접수하는 코드이다.

```
cout << "Enter the date in American format"
  << "(e.g., December 29, 1953):";
string Date;
GetLine(cin, Date, '\n');
```

월을 얻기 위하여 날자로부터 월을 분리하는 공백이 존재하는 위치를 발견하는데 find()함수를 리용하며 이 부분문자렬을 꺼내는데 Substr를 리용한다. 이러한 동작은 다음과 같은 코드로 수행된다.

```
int i = Date.find("");
string Month = Data.Substr(0, I);
```

날자의 위치를 얻어 날자를 입력하기 위해서는 날자가 있는 반점의 첫 위치를 얻어야 한다. 이 위치와 공백이 놓인 위치를 리용하여 날자에서 날의 문자수를 계산하고 그것을 입력한다. 아래의 코드는 알고리듬의 3단계를 서술한것이다.

```
int k = Date.fine(",");
string Day= Date.Substr(I + 1, k - I -1);
```

알고리듬 4 단계는 반점으로부터 문자렬의 끌사이의 두 위치에서 부분문자렬을 얻는 방법으로 년을 얻어 내는 과정이다.

```
string Year = Date.Substr(k + 2, Date.size());
```

마지막단계는 새로운 형식으로 날자를 창조하고 표시하게 하는것이다. 적당한 지령문을 써서 날자의 요소들에 대하여 련결을 진행한다.

```
string NewDate = Day + "" + Month + "" + Year;
cout << "Original date:" << Date << end1;
cout << "Converted date:" <<NewDate << end1;
```

프로그람 3-4는 완성된 프로그람이다. string클라스의 많은 기능들중 일부와 몇개 동작들만 취급하였다. 부록 3에 string클라스에 대하여 구체적으로 소개하였다.

```
// 프로그람 3-4.
#include <iostream>
#include<string>
using namespace  std;
int main() {
```

```
    //지령대기문에서 날자얻기
    cout << "Enter the date in American format"
      << "(e.g., December 29, 1953) :";
    string Date;
    getline(cin, Date, i\n i);
    // 첫 공백문자를 발견하여 월얻기
    int i = Date.find(" ");
    string Month = Date.Substr(0, i);
    // 반점문자를 발견하여 날자의 얻기
    int k = Date.find(".");
    string Day = Date.Substr(i + 1, k-i-1);
    // 반점의 뒤에 있는 공백으로부터 문장의 끝까지의 부분문장을 얻어서 년을 얻는다.
    string Year = Date.Substr(k + 2, Date.size());
    string NewDate = Day + "" + Month + "" + Year;
    cout << "Original date:" <<Date << end1;
    cout << "Converted date:" << NewDate << end1;
    return 0;
}
```

프로그람 3-4. 아메리카형식으로부터 국제적형식으로 날자를 변환

문 제

15. string 클라스를 사용하자면 프로그람에 무엇이 포함되여야 하는가?
16. 하나의 지령으로부터 하나의 문장을 읽기 위한 string 서고의 함수이름은 무엇인가?
17. 다음의 코드의 결과는 무엇인가?

```
string Message = "Volleyball!";
Message = "!!";
cout <<Message << end1;
```

18. 다음의 코드토막결과는 무엇인가?

```
sting Time = "1:42" ;
string AM = "AM";
cout << Time + AM << end1;
```

19. 다음의 코드의 결과는 무엇인가?

```
string Time = "11:15";
Time += "PM";
cout << Time << end1;
```

20. 다음 코드의 결과는 무엇인가?

```
string s = "";
```

```
cout << s.size() << end1;
```

21. 다음의 코드토막결과는 무엇인가?

```
string s="Go Wahoos!";
cout << s.size() << end1;
```

22. 다음의 코드토막결과는 무엇인가?

```
string s = "Beam Me Up Scotty";
cout  << s.Substr(5, 2) << end1;
```

23. 다음의 코드토막결과는 무엇인가?

```
string s = "The Picardmaneuver";
cout << s.Substr(4, s.size() -1) << end1;
```

24. 다음의 코드토막결과는 무엇인가?

```
string s="The Picard Maneuver";
cout << s.Substr(4, s.size() -1) << end1;
```

25. 다음의 코드토막결과는 무엇인가?

```
string s = "You will be assimilated";
int I = s.find( ", 0);
cout << s.Substr(I, s.size()) << end1;
```

26. 다음의 코드토막을 고찰하시오. 입력이 다음과 같을 때 출력결과는 무엇인가?

```
string Message;
Getline(cin, message, I, I);
cout << "Message is " << Message << end1;
Spock, you laughed, you laughed!
```

27. cin으로부터 mm/dd/yy형태로 날자를 읽어 cout에 의해 다음과 같이 날자를 출력하는 프로그람을 작성하시오.

```
Month: m
Day: dd
Year: yy
```

28. cin 으로부터 하나의 값주기명령(실례로 a=b+c)을 읽어 내는 프로그람을 작성하고 cout 에 의해 값주기명령의 왼쪽과 오른쪽을 출력하시오.

3.9 EzWindows

C++는 많은 객체들을 제공하고 있지만 사실 과제에 대하여 객체형태들을 수동적으로 작성할수 있게 하는것으로 하여 더욱 강력한것으로 되였다. 실례로 1장에서 우리는 벌레잡이유희를 실현하는 Bug라고 부르는 새로운 형태의 클라스를 창조하였다.

우리는 이전에 설계되고 수행된 객체의 사용자정의클라스들을 리용하여 프로그람을 작성할수 있었지

만 새로운 클라스들을 어떻게 창조하는가를 론의하지 않았다.

우리는 대부분 창문에서 사용할수 있는 도형표시능력을 리용하여 프로그람을 작성할수 있도록 설계된 객체들을 포함하고 있는 EzWindows 서고를 리용할것이다.

프로그람작성자들이 초기에 정의한 클라스는 SimpleWindow 와 RectangleShape 이다.

3.9.1 SimpleWindow 클라스

SimpleWindows 는 우리가 도형객체를 표시하기 위하여 창조한 창문클라스의 하나이다. Label 과 Rectangle 은 SimpleWindow 창문에서 표시될수 없다. SimpleWindow 는 아래와 같은 속성을 가진다.

- 창문의 꼭대기에 표시되는 본문
- 창문의 너비와 높이

SimpleWindow 객체가 인식하는 통보문들과 행동들은 Open 과 Close 이다. Open 통보문은 화면상에 창문를 나타나게 하여 사용할수 있게 하고 그안에 객체도 표시할수 있도록 한다. Close 통보문은 창문이 닫기고 화면에서 그 화상이 없어 지도록 한다. 그러한 실례로 Open 통보문을 SimpleWindow 에 전송하는 C++지령은

```
W.Open();
```

이다. 이 경우 Open 통보문은 임의의 변수도 요구하지 않는다.

실례코드토막

```
SimpleWindow W;
W.Open();
```

은 그림 3-4 에 보여 준 창문를 창조하고 표시한다.

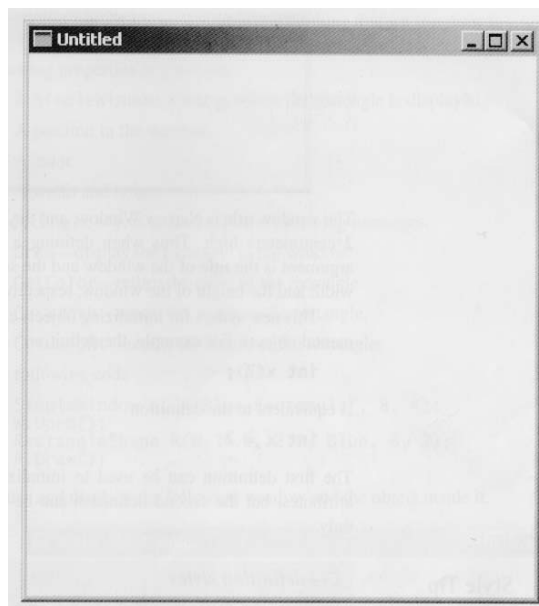
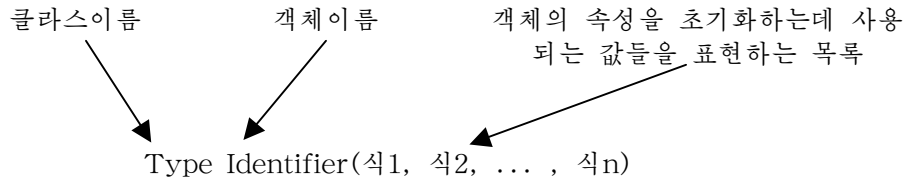



그림 3-4. 가상속성을 가진 SimpleWindow

W 에 대하여 전혀 초기화를 진행하지 않은것이 무척 놀라울것이다. 그러나 실지 W 는 초기화되였다. 클라스를 설계할 때 대부분의 프로그람작성자들은 정의한 클라스형객체의 속성에 대하여 초기화를 진행

하지 않으면 초기값들을 암시적으로 지적할수 있도록 한다. 정의한 전통적인 객체들을 초기화하기 위하여 여러가지 문법을 사용하여 객체를 정의하고 여러가지 속성값들을 줄수 있다.

그 리유는 매개의 속성들에 대하여 초기값을 다 지적하여야 하기때문이다. 하나의 객체를 정의하고 그 속성을 명백하게 초기화하는 문법은 아래와 같다.

실례로 SimpleWindow 클라스에 대하여

```
SimpleWindow N("Narrow Window", 8, 2 );
N.Open( );
```

라고 정의하면 아래와 같은 창문이 표시된다.

창문의 표제는 NarrowWindow 이고 창문의 폭은 8cm 이며 높이는 2cm 이다. 따라서 SimpleWindow 객체가 정의될 때 첫 변수는 창문의 표제이고 두번째와 세번째 변수는 창문의 폭과 너비이다. 객체를 초기화하는 새로운 문법은 역시 전통적인 객체들의 초기화에서도 리용될수 있다.

실례로

```
int  X ( 2 ) ;
```

로 정의하는것은

```
int  X = 2 ;
```

과 같다.

처음의 정의는 여러가지 속성으로 하나의 복잡한 객체를 초기화하는데 사용할수 있고 두번째 정의는 오직 하나의 단순한 객체를 초기화하는데 사용할수 있다.

알림

C++의 정의형태들

아래에서 보게 되는 정의형태들은 프로그람작성자가 작성한 객체와 전통적인 객체를 둘 다 사용할수 있는 새로운 정의이므로 우리의 모든 객체들에 대하여 이 형태의 정의방법을 사용할것이다. 이 규정은 +로 정의한 단일한것으로 발전하였다. 그러나 새로운 형태의 정의는 객체들을 하나의 표현값으로 초기화하는 경우에는 전통적인 객체에 대하여 혼란을 가져 올수 있다. 프로그람 3-2 에서 정의한것을 다시 쓰면

```
double Molecules((Mass / Formulawght) * AvogadroNmbr);
```

이다. 이 정의는 이전의것보다 리해하기 대단히 힘들어 보인다. 전통적인 객체들에 대해서는 정의형태 =를 사용하며 초기화를 요구하는 하나이상의 속성을 가진 복잡한 객체들에 대하여서는 새로운 형태의 정의를 사용하여야 한다. 복잡한 객체의 정의는 길수 있으므로 여러 행으로 분할된 정

의형식을 사용하여야 한다. 우리의 규정은 변수들사이에 정의문을 갈라 놓는것이다. 아래의 정의는 이러한 형식을 보여 준다.

```
SimpleWindow Display("A Window for Display",
  DisplayLength , DisplayWidth);
```

3.9.2 RectangleShape 클라스

하나의 창문에 여러개의 그림을 그려야 한다. 사용하기 쉬운 객체의 한가지 실례가 Rectangle 이다. 클라스 Rectangle 은 아래의 속성을 가진다.

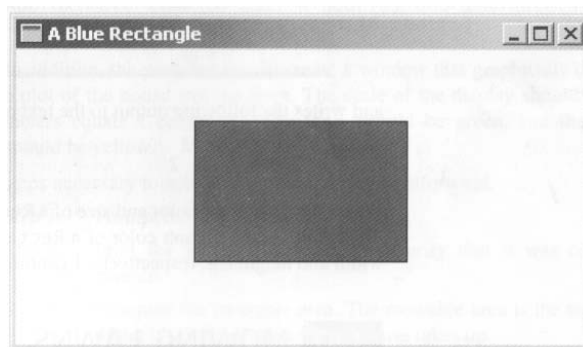
SimpleWindows 객체에 직 4 각형을 그린다.
창문에서의 위치
색
폭과 너비

Rectangle 객체는 아래의 통보문을 준수한다.

Draw : 창문에 직 4 각형을 그린다.
GetColor : 그 직 4 각형의 색갈을 돌려 준다.
GetWidth :그 직 4 각형의 폭을 돌려 준다.
GetHeight : 그 직 4 각형의 너비를 돌려 준다.

코드

```
SimpleWindow W("A Blue Rectangle", 8, 4);
W.Open();
RectangleShape R(W, 4.0, 2.0, Blue, 3, 2);
R.Draw();
```

는 아래의 창문과 그안의 객체들을 창조하고 표시한다.

R 의 정의는 창문의 왼쪽으로부터 4cm 그리고 창문의 꼭대기에서 부터 2cm 에 있는 푸른색의 4 각형을 의미한다. R 의 폭이 3cm 이고 높이가 2cm 이다. 즉 R 는 SimpleWindow W 의 중심에 놓인다.

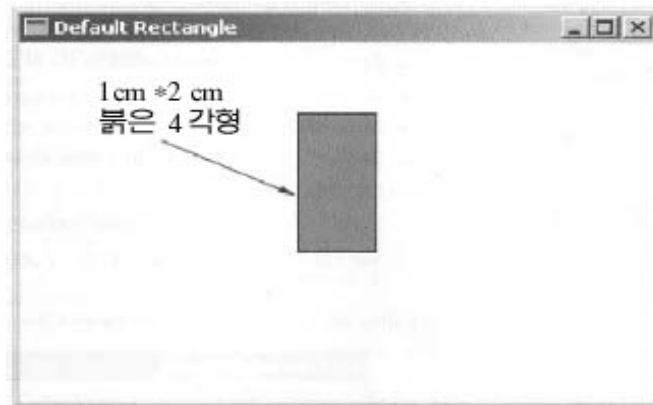
객체에 대한 정보를 얻자면 통보문을 사용하여야 한다. R 객체에 대한 실례

```
int Width = R.GetWidth ( ) ;
```

는 RectangleShape R 에 폭을 돌려 줄것을 요구하는 통보문을 전송한다. 그 값은 int 형객체 Width 에 넣어 진다.

또 하나의 다른 실례로서 4 각형을 기정폭과 너비로서 표시하는 프로그람을 작성해 보자. 코드토막

```
SimpleWindow W ("Default Rectangle",   8 ,   5 ) ;
W.Open( );
RectangleShape  D ( W,  4.0 ,  2.0  );
D.Draw( );
cout  << "Dis  width is " << D.GetWidth( )  <<  end1;
cout  <<  "Dis height is " << D.GetHeight ( ) << end1;
```

은 아래의 창문를 표시하고 다음과 같은 본문을 창문에 출력한다.

```
Dis  width  is 1
Dis  height  is 2
```

정의에서 RectangleShape 의 색과 크기를 주지 않았을 때 RectangleShape 의 초기값은 적색이고 너비와 높이는 1cm 와 2cm 이다.

3.10 잔디베기

잔디베기봉사를 하여 한시간에 6 딸라를 벌려고 하는 경우 잔디베기의 값이 얼마인가를 계산하는 프로그람을 작성하자. 이 문제는 직 4 각형잔디밭의 길이와 폭(1) 그리고 잔디밭에 위치한 집의 길이와 폭(2)을 프로그람적으로 입력하고 (모든 입력은 m 단위로 한다. 잔디베기의 평균속도는 1 초에 $1m^2$ 이라고 본다) 봉사비를 계산한다. 이 평균값에는 물공급정지량과 잔디 베는 기계의 연유공급량이 포함된다.

축적된 량은 딸라와 펜스로 인쇄될것이다.

주의

EzWindow 서고의 사용

이 책의 대부분의 프로그람들에서는 EzWindow 서고에 포함된 도형객체를 사용한다. EzWindow 서고는 대부분의 Windows 체계의 탁상화면상태에서 프로그람개발을 목적으로 제공된다. 이전의 부분에서 소개된 클라스 SimpleWindow 와 RectangleShape 는 EzWindow 서고의 포함된 부분이다. 이 클라스들을 사용하는 실행형프로그람들을 창조하기 위하여서는 알맞는 머리부파일(실례로 rect.h)을 프로그람에 포함시켜 EzWindow 서고와 련결하여야 한다.

프로그람과의 련결처리는 사용자가 사용하는 콤파일러와 조작체계에 의존한다. EzWindow 를 사용하는 여러가지 콤파일러들과 조작체계들을 위한 실행형프로그람들을 작성하기 위한 방향프로그람들은 이 책에 부속된 CD-ROM 에서 찾아 볼수 있다.

입력과 출력은 아래와 같다.

```
Please use meters for all input
Please enter the length of the lawn:150
Please enter the width of the lawn:100
Please enter the length of the house:25
Please enter the width of the house:20
Yard size:150 by 100 meters
House size: 25 by 20 meters
Approximate time to cut: 4 hour(s) and 1 minute(s)
Cost to cut: 24 dollar(s) and 16 cent(s)
```

추가적으로 집과 잔디밭을 표시하는 하나의 창문를 창조한다. 표시하는 척도는 100m 당 1cm 이다. 잔디밭은 풀색이고 집은 노란색이다.

문제를 풀기 위한 단계는 다음과 같다.

단계 1. 대기하였다가 입력자료를 읽는다.

단계 2. 사용자가 정확히 입력하였는가를 확인하기 위하여 입력자료를 인쇄한다.

단계 3. 잔디베기할 면적을 계산한다. 잔디베기할 총 면적은 잔디밭의 총 면적에서 집이 놓인 구역을 덜어 낸다.

단계 4. 잔디밭을 베는데 필요한 면적을 계산한다.

단계 5. 잔디베기를 하는데 따라 축적되는 돈을 계산한다.

단계 6. 잔디밭과 집을 보여 주기 위한 표시를 진행한다.

첫 단계를 수행하기전에 임의의 상수를 정의하여야 한다. 필요하면 탐색과 수정이 쉽도록 이 프로그람들을 묶음화하여야 한다. 아래의 상수들이 있어야 한다.

```
//Mowing rate in meters per second
const float MowRate=1.0;
//Pay rate desired
const float PayRate=6.0;
//Seconds in a minute
const in SecondsPerMinute=60;
//Seconds in an hour
const int SecondsPerHour=SecondsPerMinute*60;
//Length and width of display window
const int DisplayWidth=20;
const int DisplayHeight=20;
//Scale factor for display:100 meters equals
//1 centimeter
const float ScaleFactor=0.01;
```

프로그람의 마지막단계는 도형표시이다. 대기문에 따라 입력을 정확히 한다. 그 코드는

```
cout<< "Please use meters for all input\n" <<endl;
int LawnLength; //Length of the lawn in meters
cout << "Please enter the length of the lawn:" ;
cin >> LawnLength;
int LawnWidth; //Width of the lawn in meters
cout << "Please enter the width of the lawn:";
cin >> LawnWidth;
```

이다. 량쪽코드토막이 객체에서 서술할 이름들을 사용한다는데 주의를 돌리시오. 입력표시코드는 정확히 되였다.

```
cout << endl;
cout << "Yard size:" <<LawnLength << "by"
  << LawnWidth <<"meters" << endl;
cout <<"House size:" << HouseLength << "by"
  <<HouseWidth << "meters" <<endl;
```

다음단계는 잔디베기할 면적을 계산한다. 아래의 코드에 의하여 이 과제가 수행된다.

```
int MowableArea=(LawnLength*LawnWidth)
  -(HouseLength*HouseWidth);
```

Mowable 와 MowRate 를 사용하여 우리는 잔디베기에 필요한 시간을 계산한다. 우리는 초를 단위로 잔디베기시간을 계산한다. 5단계는 수행한 일의 총량을 현시하고 계산하는것이다.

료금을 계산하기 위하여 지불가격을 변화시킨다. 그것은 시간당 가격, 초당 가격이다. 료금계산은 다음과 같다.

```
float DollarCost = MowTimeInseconds
  *(PayRate/SecondsPerHour);
```

그리고 출력계산값은

```
int dollars = Dollarcost;
int Cents = (Dollarcost - Dollars)*100;
cout << "Cost to cut: " <<Dollars << "dollar(s)"
  << "and" << Cents << "cent(s)" << end1;
```

이다.

딸라성원은 DollarCost 의 할당에 의해 얻어 지는데 그것은 **float** 형이며 Dollars 는 **int** 형이다. 류점수형태에서 옹근수형태로 값주기하는것은 단축된 값을 저장하기 위한것이다. Dollarcost 와 Dollars 를 리용하여 펜스성원이 계산된다.

프로그람의 마지막단계는 그라프적인 도형을 현시하는것이다. 이전의 실례에서처럼 첫 단계는 구체례로 제시하고 하나의 창문을 여는것이다. 다음의 코드는 Lown 과 HousePlot 제목을 가진 Display 라는 창문을 구체례로 제시한다.

```
SimpleWindow Display("Lawn and House Plot",
```

```
    DisplayWidth , DisplayHeight);
  Display.Open();
```

창문의 너비는 DisplayWidth 로 표현된 값이며 높이는 DisplayHeight 로 표현된다. 요구대로 표시하기 위하여 처음에는 잔디밭을, 다음에는 집을 그린다. 다시말하면 큰 직 4 각형우에 작은 4 각형을 덧놓는것이다. 그러면 그것이 보이지 않을것이다. 척도화된 잔디밭의 화상을 표시하는 코드는 다음과 같다.

```
  RectangleShape Lawn(Display, DisplayWidth / 2.0,
    DisplayHeight/2.0, Green, LawnLength,
    ScaleFactor, Lawnwidth * ScaleFactor);
  Lawn.Draw();
```

잔디의 위치특성은 DisplayHeight/2.0 과 DisplayWidth/2.0 으로 되며 창문의 중심에 놓이게 된다. 최종단계는 적당한 위치에 집을 표시하는 4각형을 그리는것이다. 이 코드는 이전의 코드와 류사하다.

```
  RectangleShape House(Display, Displaywidth / 2.0,
  DisplayHeight/2.0, Yellow, HouseWidth * ScaleFactor,
  HouseHeight * ScaleFactor);
  House.Draw();
```

프로그람의 최종동작은 열려 진 창문을 닫는것이다.

```
  cout << "Type a character followed by a\n"
    << "return to remove the display and exit" <<end1;
  char Anychar;
  cin >> Anychar;
  Display.Close();
```

우의 명령문은 표준입출력창문에 하나의 통보문을 표시하고 되돌림을 허락하는 사용자의 건누르기를 기다린다. 한 문자를 써넣을 때까지 프로그람은 대기하고 있으며 창문은 없어 지지 않는다. 사용자가 한문자를 써넣으면 귀환되여 프로그람은 계속 실행되며 창문에 닫기통보문을 전송한다. 프로그람 3-5 에 완성된 코드를 주었으며 그림 3-5 는 프로그람이 창조한 창문을 보여 준다. 이 프로그람에서 몇가지 중요한 점들이 있다. 첫째는 포함명령 #include "rect.h"는 프로그람에 파일 Rect.h 를 포함시킨다. 이 파일은 프로그람작성자가 정의한 형태인 SimpleWindow 와 RectangleShape 를 사용하고 호출하게 하는데 필요한 정의를 포함한다. .h 파일을 포함하는것은 객체의 클라스와 새로운 형태를 호출하는 표준방법이다.

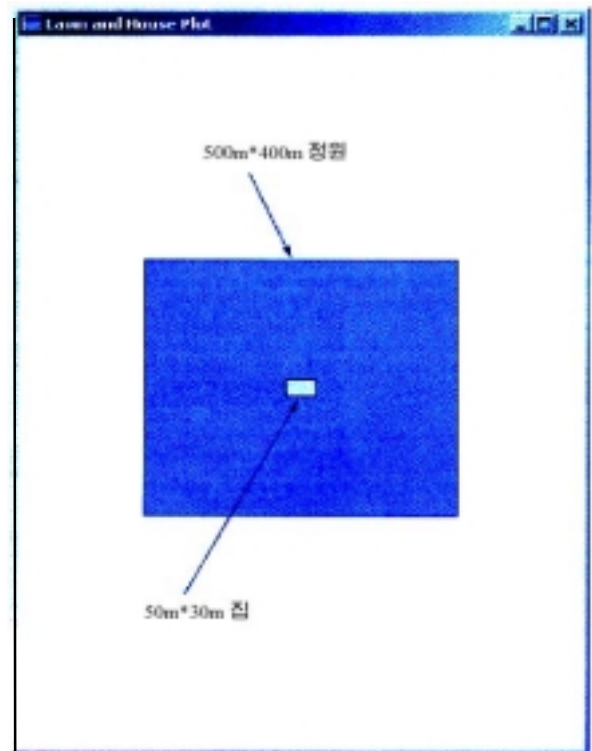



그림 3-5. 집과 잔디의 도형적인 묘사

```
// 프로그람 3-5. 잔디를 베는데 드는 료금과 시간계산
#include <iostream>
#include <string>
#include "rect.h"
using namespace  std;
int ApiMain() {
    // 초당 m2의 속도로 잔디를 베기
    const float MowRate = 1.0;
    // 요구되는 속도를 보상한다.
    const float  PayRate = 6.0;
    // 1분내의 초
    const int SecondsPerMinute = 60;
    // 1시간내의 초
    const int SecondsPerHour = SecondsPerMinute * 60;
    // 창문의 길이와 너비
    const int DisplayWidth = 20;
    const int Display Height = 20;
    // 현시하려는 척도요소
    // 100m는 1cm와 같은것으로 본다.
    const float ScaleFactor = 0.01;
    cout << "Please use meters for all input\n" << end1;
    int LawnLength; //m단위로 잔디의 길이
    cout << "Please enter the length of the lawn:";
    cin >> LawnLength;
    int Lawnwidth; //m단위로 잔디의 너비
    cout << "Please enter the  width of the lawn:·;
    cin >> LawnWidth;
    int HouseLength;
    cout << "Please enter the length  of the house:·;
    cin >> HouseLength;
    int HouseWidth;
    cout << "Please enter the length  of the house:·;
    cin >> Housewidth;
    cout << end1;
    cout <<"Yard Size:" << LawnLength << "by"·
      << LawnWidth << "meters" << end1;
    cout << "HouseSize: "<<HouseLength << "by"·
      << HouseWidth << "meters" << end1;
    int MowableArea = (LawnLength * Lawnwidh)
```

```
        - (HouseLength * HouseWidth);
    int MowTimeInseconds = MowableArea / MowRate;
    int Hours = MowTimeInSeconds / SecondsPerHour;
    int Minutes = (MowTimeInSeconds % SecondsPerHour)
        / SecondsPerMinute;
    cout << "Approximate time to cut :" << Hours
        << "Hour(s)" << Minutes << "minute(s)" << end1;
    float Dollarcost = MowTimeInSeconds * PayRate
        / SecondsPerHour;
    int Dollars = DollarCost;
    int Cents = (DollarCost - Dollars) *100;
    cout << "Cost to cut:" << Dollars << "Dollar(s)"
        << "and••0" << Cents << "cent(s)" << end1;
    SimpleWindow Display(Lawn and House Plot•,
    DisplayWidth, DisplayHeight);
    Display.Open();
    RectangleShape Lawn(Display, DisplayWidth / 2.0,
    DisplayHeight / 2.0. Green, LawnLength * ScaleFactor,
        LawnWidth *ScaleFactor);
    Lawn.Draw();
    RectangleShape House( Display, DisplayWidth/2.0,
        DisplayHeight/2.0,  Yellow, HouseLength *ScaleFactor,
        HouseWidth * ScaleFactor);
    House.Draw();
    cout << "Type a Character followed by a\n "
        << "return to remove the display and exit" << end1;
    char Anychar;
    cin >> Anychar;
    Display.Close();
    return 0;
}
```

프로그람 3-5. 잔디를 베는데 드는 시간과 비용계산

함수 main()이 이 프로그람에는 없지만 코드는 ApiMain()라는 함수에 포함되여 있다.

창문안에 표시된 객체를 관리하고 이러한 창문을 작성하기때문에 C++콤파일러가 제공하는 일반창문 창조기교를 리용하여야 한다.

우리가 이 특별한 기능을 요구하는 원인은 도형프로그람기교를 론의하는 10 장에서 설명이 구체적으로 되기때문이다. 그래서 지금 도형표시객체에 대하여 하나의 분리된 창문을 창조하는 프로그람을 작성할 때 프로그람의 시작점이 Main()이 아니라 ApiMain()이라고 부르는 함수의 실행으로부터 프로그람이 시작하는것이다.

프로그람을 주의깊에 관찰해 보면 2 개의 중요한 문제들이 생긴다. 잔디의 면적이 척도화된것이 창문보다 더 크면 어떤 현상이 나타나는가? 또한 집의 크기가 밭의 면적보다 크다면 프로그람실행시에 무엇이 나타나는가?

물론 명백하게 프로그람을 실행하고 그 동작을 고찰해 볼수 있지만 이러한것을 무시할수 있다. 프로그람의 호상작용에서 입력동작은 아주 중요하다.

검사의 이러한 형태를 수행하는 기교는 아직 없지만 다음장에서는 사용자의 입력에 의하여 이러한 검사기능을 수행하는 여러가지 C++기능을 보여 준다.

문 제

29. SimpleWindow 객체 Try 를 창조하는 C++정의를 쓰시오. 제목띠에는 "Good Job!"라고 씌여 져 있으며 창문의 너비는 6cm, 높이가 3cm 이다.
30. 제목이 "Nice Job"인 SimpleWindow 를 창조하는 프로그람을 작성하고 창문의 중심위치에 하늘색 4 각형을 그리시오. 창문은 너비가 5cm, 높이가 10 이며 4 각형은 너비가 3cm, 높이가 2cm 이다.
31. RectangleShape 의 너비값을 얻기 위한 통보는 무엇인가?
32. RectangleShape 의 높이를 얻기 위한 통보는 무엇인가?
33. 다음의 코드토막에 의하여 그려 지고 창조된 4 각형을 설명해 보시오. 4 각형은 크기가 얼마이며 무슨 색인가?

```
SimpleWindow T(Check Your knowledge, 10, 10);
T.Open();
Rectangle P(T);
T.Draw();
```

콤퓨터의 력사

기교장치들

17 세기 초기에 기계식계산기가 발명되였다. 첫 기계적인 계산기의 조립과 함께 유명해 진 사람은 윌헬름 스키카르다(1592~1635)이다.

《계산하는 시계》라고 불리우는 이 장치의 가장 중요한 혁신의 하나는 자리올림을 위한 맞물림기구를 사용한것이다(그림 3-6). 이 기계는 후에 많은 기계계산장치들에서 여러가지 형태로 리용되였다. 이 장치의 발명에 대한 상세한 명세서는 그후 수학가이며 천문학자인 요한네스 케플레르에게 보낸 편지가 발견된 1960 년대까지 알려 지지 못하였다.

그후 다른 계산기의 발명가 파스칼이 많은 주목을 끌었다. 파스칼(1623-1662)은 어렸을 때부터 머리가 비상한 사람이였다. 어린 나이에 그는 여러가지 수학적인 정의들과 여러개의 가설들을 제기하였다. 30 년의 길지 않은 한생을 마친 그는 싸이폰과 수압프레스를 발명했다.

파스칼은 그림 3-7 에서 보여 주는 장치를 설계하고 제작하였다. 이 장치는 여러개의 이발이 있는 치차를 가진 하나의 작은 통이였다. 매 치차는 하나의 수자를 표현하고 그 수자는 하나의 작은 창문으로 들여다 보이는 치차바퀴우에 표시되였다. 더하기는 눈금에 새긴 수자들이 씌여 있는 치차의 회전에 의하여 수행되고 치차우의 창문들에서 결과를 읽었다. 그 기계도 좀 복잡하였지만 덜기연산도 진행하였다. 이것은 간단한

장치이지만 유명한 파스칼의 계산장치로 되였다. 기계장치를 생산하기 위하여 파스칼은 여러가지 어려운 문제들을 풀어야 하였다.

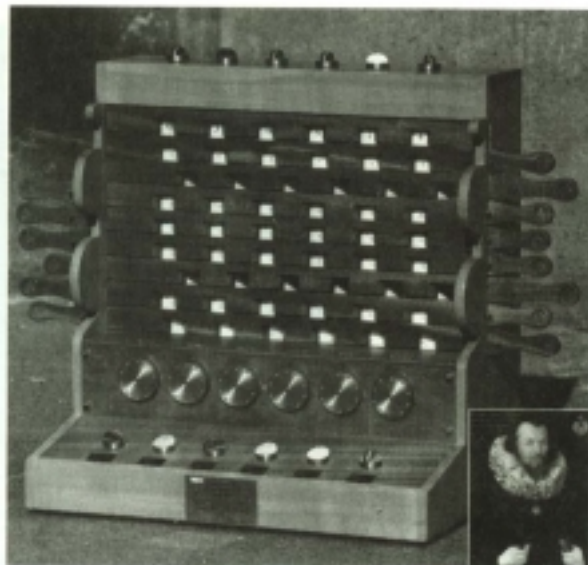



그림 3-6. 윌헬름 스키카르다의 《계산하는 시계》

윌헬름 레비네츠는 재능 있는 사람이였다. 그는 론리학, 수학, 물리학, 리론력학부분에 종사하였다. 그는 여러가지 계산을 할수 있는 독자적인 장치를 개발하여 잘 알려 졌다. 그는 뉴톤에 뒤이어 효과적이지는 못하지만 20 년을 연구하여 수판을 발명하였는데 이 수판을 단계식계산장치(Stepped reckoner)라고 하였다. 치차는 원통에 수직으로 9 개의 이발을 가지고 있었다. 적당한 사이를 두고 불안정을 조절하면서 치차는 하나의 10 진수를 왼쪽으로 넘기였다. 레비네츠는 파스칼과 마찬가지로 수판을 만들기 위하여 요구되는 기술적문제를 혼자서 풀었다.

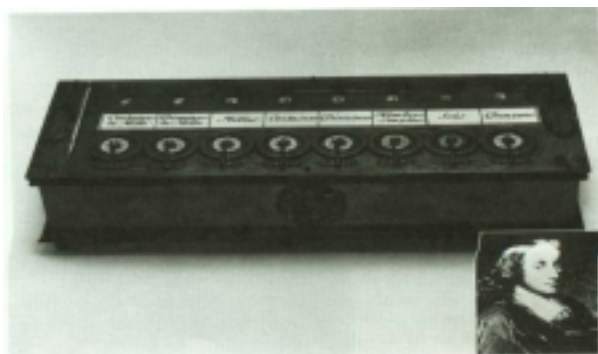



그림 3-7. 블래즈 파스칼과 그가 발명한 계산장치

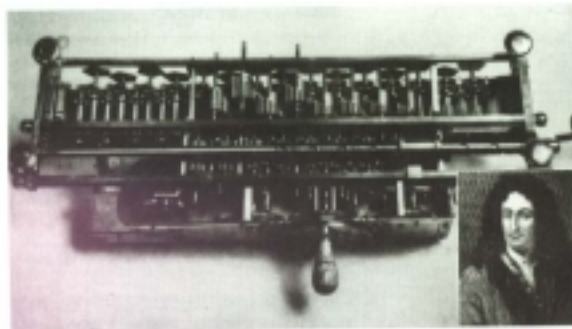



그림 3-8. 윌헬름 레비네츠와 단계식계산장치

3.11 알아 둘 점

✓ 객체는 선언할 때 초기화된다.

✓ 류점수값은 옹근수객체에 보관될 때 옹근수값으로 변화된다. 실례로 다음의 코드가 실행되면 x 의 값은 23 이다.

- int x = 0;
- x = 23.6;

✓ 값주기연산자는 산수연산자보다 우선권이 낮다.

✓ C++예약어 const 는 변경할수 없는 객체를 정의하는데 리용된다. 이 기능은 물리적인 상수를 표현하는 값을 유지하도록 객체를 정의하는데 아주 효과적이다. 이 값은 프로그람의 실행시에 변할수 없다.

✓ 입출력지령객체 cin 은 입력지령이다. 이 지령은 건반에서의 입력을 받아 처리한다.

✓ 입력은 입력연산자 >>을 사용하여 하나의 입력지령으로부터 얻는다. 입력연산자는 입력지령으로부터 int 형, float 형문자값들을 입력한다.

✓ 사용자로부터 자료를 입력받는 프로그람을 작성할 때 프로그람은 입력을 요구하는 지령재촉문과 입력형식을 지정해야 한다.

✓ 프로그람의 중요한 특징은 정확하고 알기 쉬워야 한다는것이다.

✓ C++표준서고는 대단히 우수한 클라스들을 포함하고 있다. C++에 정통하려면 어떤 특징이 있고 그것들을 어떻게 사용하는가를 알아야 하는것이다.

✓ string 클라스는 문자렬을 보관하거나 문자렬의 순서를 달리하기 위하여 설계되였다.

✓ #include <string>은 string 서고에 제공된 기능을 호출하기 위하여 반드시 필요하다.

✓ 문자렬은 다른 문자렬이나 문자렬상수로서 초기화될수 있다. 문자렬은 한 문자로 초기화될수 없다.

✓ 문자렬객체에 포함된 문자렬은 끝문자 '\0'을 포함하지 않는다.

✓ 련결은 2 개의 문자렬을 《붙여》 하나의 새로운 문자렬을 창조하는 처리이다. +연산자는 련결연산자이다. 실례로 FirstName 과 LastName 문자렬을 창조하는 코드는 다음과 같다.

```
Fullname = firstName + i+ LastName;
```

✓ 2 개의 문자렬을 더할 때 +=연산자는 추가연산을 수행한다. 아래에 Year 문자렬을 date 문자렬에 추가하는 명령문을 주었다.

```
Date += Year;
```

✓ +=, -=, *=, /=, %=의 값주기연산자를 제공하고 있는 C++는 하나의 객체에 대하여 하나의 산수연산을 수행하며 그 객체안에 결과값을 기억한다. 실례로 그 명령은 다음과 같다.

```
x +=5;
x = x+ 5;
```

✓ C++는 옹근수와 류점수를 증가하거나 감소시키는 ++와 --연산자를 가지고 있다. 이러한 연산자들은 오른쪽연산자와 왼쪽연산자위치에 존재할수 있다. 앞에 증가 또는 감소연산자가 있을 때 그 값은 변경되기전의 객체값을 나타낸다. 그 실례는 다음과 같다.

```
int x = 10;
int i = x++;
```

이것을 실행하면 i 의 값은 10 으로 설정되고 x 는 11 로 된다. 앞의 증가 또는 감소연산자에 대한 표현값은 객체가 증가된 값이다. 실례로 코드토막

```
int x = 9;
int i = -x;
```

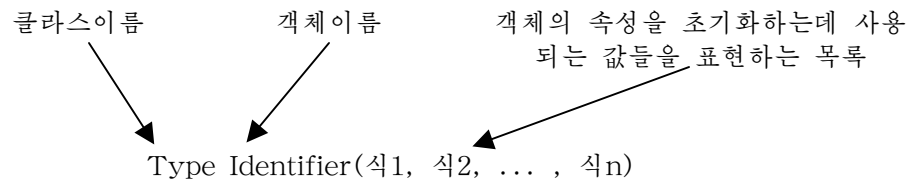
를 실행하면 i 값은 8 로 되며 x 값은 8 로 된다.

✓ 객체에 통보문을 전송하는 C++문법은 다음과 같다.

✓ string 클라스는 문자렬의 크기를 얻는 기능들을 제공하는데 문자렬에서 문자를 입력하고 문자렬에 출력을 할수 있는 기능도 가지고 있다.

✓ 이 행에서는 Getline()을 리용하여 문자렬을 읽을수 있다. 다음의 코드는 문자렬 studentRecord 에서 cin 으로부터 입력한 행을 읽는다.

```
Getline(cin, StudentRecord, i\ni);
```

✓ 여러개의 값 혹은 속성을 가진 객체는 특수한 문법을 리용하여 정의되고 초기화된다. 그 문법은 다음과 같다.

련습문제

3.1 C++의 >>연산자의 이름은 무엇인가?

3.2 다음의 코드가 왜 틀리는가를 설명하시오.

```
int x = 5;
int y;
int z;
(z = y) = x;
```

3.3 다음의 코드가 왜 틀리는가를 설명하시오.

```
const float PayRate = 6.50;
OldSalary = PayRate *40;
PayRate = 7.25;
NewSalary = PayRate * 40;
```

3.4 *=, +=, -=, /=. %=의 이름은 무엇인가?
3.5 흐름 cin 이 리용되기전에 무슨 파일이 선언되여야 하는가?
3.6 다음의 식에서 객체 Cost 와 Price 의 최종값을 쓰시오.

```
int Cost = 5;
int Price = 10;
++Cost;
Cost++;
Cost = Price++;
Cost = ++Price;
Cost  = Price++ + ++Price;
Cost +=Price ;
Cost *=5;
Cost += Price *5;
++Cost++;
cost = ++Price++;
Price /= Cost ++
```

3.7 2 개의 명령문을 하나의 명령문으로 만드시오.

```
j += 1;
i = j;
```

3.8 2 개의 명령문을 하나의 명령문으로 만드시오.

```
i = j;
j += 1;
```

3.9 원둘레를 계산하기전에 프로그람 3-1 에서 π의 값을 3.1415926 으로 변경하는 명령문을 추가하시오. 수정된 프로그람을 번역하시오. 번역시 오유가 있는가? 그렇다면 왜 오유통보가 생기는가?
3.10 원의 직경을 계산하도록 프로그람 3-1 을 수정하시오.
3.11 하나의 사탕에 포함된 분자수를 계산하도록 프로그람 3-1 을 수정하시오. 사탕은 수소, 탄소, 산소를 포함하고 있다. 우리는 산소의 원자량을 고려하여야 한다. 작성하는 프로그람은 포도당 10g($C_6H_{12}O_6$)분자수를 계산하는것으로 하시오.
3.12 아래의 값주기명령에서 왼쪽의 객체에 넣어 진 값을 구하시오. 이 련습에서 **char** 는 8bit, **short** 는 16bit, **int** 는 32bit 이다. 객체를

```
class c;
short s;
int  i ;
```

로 선택하면 10 진수값으로 대답하시오.

```
i = 101.3;
i = 59.8;
s = Ox3a1;
i = 25;
i = 3e2;
```

```
c = 'a';
i = 31 + 0.7;
s = 12.2 + 13;
c = 55L;
c= Oxff2
c = 0773451;
i = 0.23'
o = Ox143F;
c = i0I;
i = i9I;
s = 31000;
```

3.13 아래의 명령이 실행된후 객체값을 구하시오. 객체의 값을 모르면 그 값은 정의되지 않은것으로 보시오.

```
int  i = 15;         double x=5.2;
int k = 10;          double y=0.0;
int j = 0;           double z=0.0;
i = k;               y=x;
k = j;               z=y;
j = i;               x=z;
int i = 5;           double x=32.1;
int j = 6;           double y=45.0;
int k = 7;           double z=0.0;
j = i;               x=z;
i = 3;               y=z;
k = j;               x=y;
int i = 21;          y=x;
int j = 0;           char a=iai;
int k = 11;          char b=ibi;
j = k;               char c=ici;
k = i;               a=b;
j = k;               b=c;
                     c=a;
```

3.14 매 값주기식에 대한 결과(<값, 형태>를 사용하여)를 구하시오. 값주기가 리용될수 없다면 <비정의, 비정의>로 결과를 구하시오.

```
char c = 'A';
int i = 23;
float x = 3.1;
double  z = 5.0;
```

결과를 10진수값으로 표시하시오.

```
i = c;            x = i = z;
i = x;            i = x = z;
i = i;            z = x = I;
x = i;            z = i = c;
z = x;            I = c = z;
x
= i = c;
```

3.15 아래의 const 정의가 옳은가를 설명하시오.

```
const x=23;
const float z=5;
const int I=5;
const double int =5;
const double x=33;
const char Blcnk(040);
const float f= +3;
const double x= 3.0;
const float z(Ze3);
```

3.16 다음의 코드토막을 보시오.

```
int NumberOffDays = 30;
float PayRate = 5.0;
float AverageHours = 5;
cin >> PayRate >> NumberOffDays >> AverageHours;
cout << "Salary is" << PayRate * NumberOfDays *
  AverageHours <<end1;
```

다음과 같은 문자가 입력될 때 출력결과를 쓰시오.

```
4.50 023 4
6.00 23 010
7.00,5,12
7.00*5*12
```

3.17 float 형보다 더 정확한 double 형을 가지는 C++의 기능을 알아 보시오. 이 기능을 리용하여 double 형이 float 형보다 더 정밀하다는것을 증명하는 프로그람을 작성하시오.

3.18 프로그람 3-1 에서 π의 값을 더 정밀하게 넣으시오. 당신은 프로그람실행시에 이러한 차이의 결과를 볼수 있는가? 왜 그런가?

3.19 목록 3-1 에 보여 준 프로그람을 주별평균, 일별평균을 계산하는 프로그람으로 수정하시오. 이 변화는 평가된 년말보관자료와 차이가 있는가? 왜 그런가?

3.20 목록 3-1 에 보여 준 프로그람이 0.5 딸라화페를 처리하도록 수정하시오.

3.21 프로그람 3-5 가 2 개의 건물이 있는 잔디밭을 취급하도록 수정하시오. 두번째 건물은 차고라고 가정하시오.

3.22 폭이 8cm, 높이 8cm 인 하나의 창문에 직선과 푸른 선의 바른 4 각형을 만드시오. 바른 4 각형은 한 변이 2cm 이다.

3.23 하나의 탑을 이루는 5 개의 4 각형을 그리는 프로그람을 작성하시오. 제일 아래의 4 각형들은 폭이 8cm, 높이가 10cm 이며 매 잇달린 4 각형은 그 아래 4 각형의 길이의 75%만한 크기를 가진다. 모든 직 4 각형의 높이는 같다. 직 4 각형은 청색이다.

3.24 빈 직 4 각형을 그리는 프로그람을 작성하시오. 직 4 각형은 높이가 크고, 폭이 좁은 2 개의 수직직 4 각형과 높이가 작고 폭이 넓은 2 개의 수평직 4 각형으로 이루어 졌다. 직 4 각형은 하나의 직 4 각형을 이루도록 끝이 서로 맞물려 있다. 직 4 각형들은 노란색이다.

3.25 지령대기문에서 ddd-ddd-ddd 형식으로 하나의 전화번호를 입력하고 그것을 (ddd) ddd-dddd 형태로 인쇄하는 프로그람을 작성하시오. 여기서 d 는 수자이다.

3.26 인치단위로 거리를 입력하는 프로그람을 작성하시오. 프로그람에서는 마일과 피트, 인치로 거리를 인쇄한다.

3.27 물의 가격을 계산하는 프로그람을 작성하시오. 입력값은 갈론단위로 평가되는 값이다. 물은 다음과 같이 계산된다.

- 물의 가격은 100 갈론당 0.0021 펜스
- 봉사비는 100 갈론소비당 0.001 펜스
- 봉사에서 2%만한 편차는 물의 총량과 봉사의 변화에 의하여 계산된다.

3.28 지령재촉문에서 류점수를 읽어 들이는 프로그람을 작성하시오. 그 프로그람은 한개 행에 옹근수 부분을 인쇄하며 두번째 행에 소수점을 인쇄한다. 실례로 프로그람의 입력값이 23.45 라면 출력결과는 다음과 같이 된다.

23, 0.45

3.29 프로그람 3-4 를 월은 세 문자로,년은 두 문자로 표시하는 국제적인 날자형식으로 출력하도록 수정하시오. 실례로 날자 March 18, 1983 은 18 Mar 83 으로 출력된다.

3.30 Tom Paris 의 형태로 하나의 이름을 입력하고 Paris, Tom 으로 출력하는 프로그람을 작성하시오.

3.31 폭이 20cm, 높이가 20cm 인 하나의 SimpleWindow 를 창조하는 프로그람을 작성하시오.

3.32 우리는 해발고가 높아 지면 공기압력이 낮아 진다는것을 알고 있다. 일정한 고도에서 정밀한 공기의 압력은 공기의 밀도와 온도와 같은 여러가지 요인들에 관계된다. 공기의 압력때문에 해발고에 따라 다른 실질적공기의 압력은 고도 8m 당 1000 분의 1 씩 떨어 진다. 바다수면에서 공기의 압력과 일정한 위치에서의 공기압력을 1000 분의 1 단위로 입력하고 그 위치에서 해발고를 m 단위로 계산하는 프로그람을 쓰시오.

3.33 hh;mm;ss 형태로 결과시간을 입력하고 초단위로 결과시간을 계수하여 그것을 출력하는 프로그람을 작성하시오. 프로그람은 아래의 시간들에 대하여 실행된다.

2:5:10
5:30:4
10:4:30

3.34 10cm 의 폭과 8cm 의 높이를 가진 Window 를 만드는 프로그람을 작성하시오. 폭이 5cm 이고 높이가 4cm 인 푸른 RectangleShape 를 그리시오. RectangleShape 의 중심은 창문의 왼쪽끝에서 3cm 이고 창문의 웃쪽에서부터 5cm 이다. 프로그람을 실행하고 동작을 검사하시오.

제 4 장. 조종구조

소개

지금까지 작성한 프로그람들은 실행하는 순서가 간단하였다. 프로그람은 첫 명령문으로부터 시작하여 마지막명령문까지 한번만 실행되였다. 이런 형식의 프로그람작성방법은 간단한 문제를 푸는데는 충분하였다. 그러나 일반적인 문제를 해결하자면 어느 명령문이 몇번 실행되는가를 조종할수 있는 능력이 필요하다. 이 장에서는 if와 switch 등 2개의 분기구조와 while, for, do와 같은 3개의 반복구조를 분석하게 된다. switch를 제외하고는 모두 론리식을 리용하는데 C++는 론리자료형(bool)을 지원한다.

기본개념

- 론리값과 연산자
- 진리값표
- bool형
- 관계연산자
- 단락(short-circuit)평가
- if-else 명령문
- switch명령문
- break명령문
- enum명령문
- for구조
- 무한순환
- while구조
- do구조
- 불변

4.1 론리대수

론리식은 론리값 《참》과 《거짓》으로 표현되는 식이다. 실례로 아래의 론리식을 분석해 보자.

- 0℃는 32°F와 같다(맞다 - 《참》).
- 3각형은 4개의 면을 가진다(틀린다 - 《거짓》).

론리값의 조작과 관련한 수학분야를 론리대수라고 한다. 론리대수는 19세기 영국의 수학자 죠지 부울이 정의하였다.

론리값과 식들은 가상검증의 기본블로크이므로 수학적으로 중요하다. 콤퓨터조작에서는 조종명령문을 리용하여 하드웨어의 동작을 모형화할수 있다. 론리값을 론리식에 조합하기 위한 기본론리연산자들은 and, or와 not이다.

4.1.1 진리값표

진리값표는 론리연산이 참으로 되는 조건과 거짓으로 되는 조건을 규정하는 표이다. 진리값표는 연산수의 가능한 조합을 모두 반영하며 매 조합에 따르는 연산결과를 모두 반영한다.

론리연산자 and(론리적)의 진리값표를 표 4-1에 주었다. 이 진리값표에서 P와 Q는 2진론리연산자의 왼쪽과 오른쪽연산수를 의미한다. 그러나 표 4-3에서 P는 단항론리연산자인 not의 연산수를 의미한다. 2진론리연산자는 4개의 론리조합을 가지므로 진리값표에 4개를 기입하여야 한다. 단항론리연산자인

경우에는 2개만을 기입한다. 진리값표는 그의 연산수가 모두 참일 때만 연산이 참이라는것을 보여 준다. 실례로 진리값표의 두번째 행에서 P가 참이고 Q가 거짓일 때 AND(론리적) 연산은 거짓이다. 4번째 행의 P와 Q가 다 참이므로 AND(론리적)연산이 참이다.

4.1.2 론리식

론리연산을 조합하여 식을 합성할수 있다. 실례로 다음의 식은 P와 Q가 다 《거짓》일 때 《참》이고 다른 경우에는 《거짓》이다.

not (P or Q)

P와 Q가 다 《거짓》일 때 보조식 (P or Q)은 《거짓》이고 이 보조식의 부정은 《참》이다. P와 Q가 《참》이면 보조식 (P or Q)는 《참》이며 이 식은 《거짓》이다. 표 4-4에 있는 진리값표는 이것을 분석한 표이다. 표에는 보조식 (P or Q)와 전체 식 not(P or Q)의 란을 다 주었다.

다음의 식은 P가 《거짓》이고 Q가 《참》일 때만 《참》이다.

(not P)and Q

이 장의 마지막에서 복습을 통하여 다른 연산자들과 드모르간의 법칙을 포함한 론리들과 법칙들을 분석한다.

표 4-1. 론리적에 대한 진리값표

P	Q	P AND Q
거짓	거짓	거짓
거짓	참	거짓
참	거짓	거짓
참	참	참

표 4-2. 론리더하기에 대한 진리값표

P	Q	P OR Q
거짓	거짓	거짓
거짓	참	참
참	거짓	참
참	참	참

표 4-3. 론리부정에 대한 진리값표

P	not P
거짓	참
참	거짓

표 4-4. not(PorQ)에 대한 진리값표

P	Q	PorQ	not(PorQ)
거짓	거짓	거짓	참
거짓	참	참	거짓
참	거짓	참	거짓
참	참	참	거짓

표 4-5. (notP)andQ에 대한 진리값표

P	Q	notP	(notP)andQ
거짓	거짓	참	거짓
거짓	참	참	참
참	거짓	거짓	거짓
참	참	거짓	거짓

4.2 론리형

C++에서 론리값을 표현하는데는 시간이 걸렸다.

C++의 이전 판에서는 C프로그람작성언어와 같은 규칙을 사용하였는데 그 규칙은 론리값 《거짓》을 령으로 표현하고 론리값 《참》을 령이 아닌 수로 표현한다. 이러한 규칙에 따라 다음의 식들

1 979 *3

은 《참》이고

다음의 식들

0 17% 17

은 《거짓》이다.

현재 C++는 론리자료형을 지원하고 있다. 그것을 bool이라고 하며 기호상수 true와 false를 대입할 수 있다. 다음의 정의를 보시오.

```
bool MoreDataToProcess = true;
```

```
bool ErrorHasBeenDetected = false;
```

이 정의를 보면 bool자료형과 그의 기호상수를 리용하여 프로그람의 분석과 리해를 쉽게 할수 있다는것을 알수 있다. 이 효과로 하여 론리적인 개념을 표현할 때 bool형객체를 리용하게 된다.

4.2.1 론리연산자

C++에는 3개의 론리연산자 즉 &&, ||, !가 있다. &&연산자는 and론리연산자를 실현하는데 리용되며 ||연산자는 or론리연산자를 실현하고 !연산자는 not론리연산자를 실현한다.

객체들을 다음과 같이 정의하자.

```
bool p = true;
bool q = false;
bool r = true;
bool s = false;
```

그러면 아래의 식들은 《참》이고

```
p        //p는 true이다.
p && r   //두 연산수가 true이므로 결과는 true이다.
p || q   //하나의 연산수라도 true이므로 결과는 true이다.
! s      //연산수가 false이므로 결과는 true이다.
```

다음의 식들은 《거짓》이다.

```
q        // q는 false이다.
p && s   // 하나의 연산수라도 true이면 결과는 false이다.
q || s   // 두 연산수가 다 false이므로 결과는 false이다.
! r      // 연산수가 true이므로 결과는 false이다.
```

bool형객체를 출력하거나 입력할 때 bool형객체는 암시적으로 2진수로 표시한다. 따라서 p가 bool형객체이면 출력명령문

```
cout << p <<end1;
```

은 p가 true인가 false인가에 따라 1아니면 0을 표시한다. 표준입력에 들어온 값이 0이면 명령문 cin>>p;은 p에 false을 할당한다. 대신 0이 아닌 값을 입력하면 명령문 cin>>p;은 p에 true를 할당한다.

론리연산자는 또한 표준자료형인 int나 char로 정의된 객체에 대해서도 적용할수 있다.

이러한 객체에 론리연산자를 적용할 때에 앞에서 언급한 규칙이 그대로 적용된다. 다음의 실례가 바로 그러하다.

```
int i = 1;
int j = 0;
int k = 0;
int r = -1;
int m = 0;
```

이러한 정의에 기초하여 다음의 식들의 결과는 true이다.

```
i        //i는 0이 아니다.
i && k   //모든 연산수들이 령이 아니다.
!j       //연산수가 0일 때 not는 참이다.
```

반면에 다음의 식들은 false이다.

```
j        //j는 0이다.
j || m   //두 연산수가 다 0이다.
!r       //연산수가 0이 아닐 때 not는 참이다.
```

4.2.2 관계연산자

론리값을 조작하는 연산자에는 론리연산자외에 관계연산자도 있다. 관계연산자에는 2종류의 관계연산자가 있다. 즉 갈기연산자와 비교연산자가 있다.

갈기연산자는 기본자료형과 지적자형(지적자형은 11장에서 보게 된다)에 대하여 적용한다. 갈기연산자에는 ==와 !=가 있다. 이 연산자들은 2개의 객체값이 같은가 다른가를 결정한다. 두개의 연산수가 같은 값을 가지면 ==연산자는 true이고 아니면 false이다. !=연산은 이와 반대이다. 즉 2개의 연산수값이 다르면 연산은 true이고 아니면 false이다. 비교연산자 역시 기본자료형과 지적자형에 대하여 적용한다. 이 연산자는 2개 값의 크기를 비교하는데 리용된다.

비교연산자에는 <, >, <=, >= 4개가 있다. <연산자는 수학에서 작기와 같다. 즉 왼쪽연산수가 오른쪽연산수보다 작으면 true이고 아니면 false이다. >연산자는 수학에서 크기와 같다. 즉 왼쪽연산수가 오른쪽연산수보다 크면 true이고 아니면 false이다. <=연산자는 수학에서 크지 않기이다. 즉 왼쪽연산수가 오른쪽연산수보다 작거나 같다면 true이고 아니면 false이다. >=연산자는 수학에서 작지 않기이다. 즉 왼쪽연산수가 오른쪽연산수보다 크거나 같다면 true이고 아니면 false이다. bool자료형에 대해서는 false가 true보다 작다. true와 false값은 특별히 정해 지지 않았다. 객체정의가 다음과 같은 경우

```
int i = 1;
int j = 2;
int k = 2;
char c = '2';
char d = '3';
char e = '2';
```

아래의 식들의 결과는 true이다.

```
c == e    //==는 2개의 연산수가 같으므로 true이다.
i != k    // 2개의 연산수가 다르므로 결과는 true이다.
i < j     // 왼쪽연산수가 오른쪽연산수보다 작으므로 결과는 true이다.
d > e     // 왼쪽연산수가 오른쪽연산수보다 크므로 결과는 true이다.
i <= k    // 왼쪽연산수가 오른쪽연산수의 값보다 크지 않으므로 결과는 true이다.
j >= k    // 왼쪽연산수가 오른쪽연산수보다 작지 않으므로 결과는 true이다.
```

또한 다음의 식들의 결과는 false이다.

```
i == j    // 2개의 연산수가 다르므로 false이다.
```

c != e   // 2개의 연산수값이 같으므로 false이다.
j < k   // 왼쪽연산수가 오른쪽연산수보다 작지 않으므로 결과는 false이다.
c > e   // 왼쪽연산수가 오른쪽연산수보다 크지 않으므로 >는 false이다.
d <= c   // 왼쪽연산수가 오른쪽연산수보다 크므로 false이다.
i >= k   // 왼쪽연산수가 오른쪽연산수보다 작으므로 >=는 false이다.

주의

값주기와 같기의 혼돈

일반적으로 프로그람작성에서 = 연산자를 잘못 리용하면 오유를 범할수 있다. 아래와 같은 2개의 식을 보기로 하자.

i == 0
i = 0

첫번째 식은 같기식이고 i가 0이면 true이다. 두번째 식은 값주기식이며 결코 true가 아니다. 왜냐하면 그 식의 값이 i에 대입된 값 0이기때문이다.

4.2.3 연산자우선권평가

여러개의 연산을 리용하는데서 복잡한 식을 작성하는 경우가 있다. 이러한 식을 평가하자면 관계연산자와 론리연산자들의 우선권에 대하여 알고 있어야 한다.

부정연산자 !는 다른 단항연산자들과 마찬가지로 높은 우선권을 가진다. 2항연산자들가운데서 관계연산자와 론리연산자는 산수연산자보다 우선권이 낮으며 값주기연산자들보다 우선권이 높다.

관계연산자는 론리연산자보다 우선권이 높다. 관계연산자가운데서 비교연산자는 같기연산자보다 우선권이 높다. 론리연산자들가운데서 &&는 ||보다 우선권이 높다. 따라서 다음의 식들은 의미가 같다.

i + 1 < j * 4 && ! p || q
(((i + 1) < (j * 4)) && (!p)) || q

그리고 다음의 식들도 역시 의미가 같다.

p != i < j || q && s
(p != (i < j)) || (q && s)

표 4-6. 연산자들의 우선권

연산자
단항연산자
* , /
+ , -
관계연산자
AND
OR
값주기연산자

4.2.4 단락평가

론리식을 평가할 때 때때로 연산수들을 다 분석하지 않고도 론리식의 값을 알수 있는 경우가 있다. 실례로 &&연산자의 한 연산수가 false라는것을 알면 &&연산의 결과가 false임을 알수 있다. 마찬가지로 ||연산의 한 연산수가 true이면 ||연산의 결과가 true임을 알수 있다.

론리연산을 평가할 때 C++는 오른쪽연산수보다 먼저 왼쪽연산수를 평가한다. 또한 연산결과가 왼쪽연산수에 의하여 결정될수 있다면 오른쪽연산수는 평가하지 않아도 된다. 이런 형의 평가를 단락(short-circuit)평가라고 한다.

단락평가는 보통 론리식에서 리용되는 객체들을 조작하기전에 진행된다. 다음의 명령문들을 분석하자.

(i != 0)) && ((j / i) > 5)

&&의 왼쪽연산수가 먼저 평가되므로 i가 0이 아니면 i로 나누기가 허용된다는것을 알수 있다. 단락평가를 하지 않으면 &&의 오른쪽연산수 i가 0으로 평가되여 나누기오유가 생기게 된다.

주의

자리올림오유

류점수에 대하여 같기연산을 리용할 때 주의해야 할 점이 있다. 류점수자료형의 유한정확도를 가진 반복연산에서 반올림오유가 생긴다. 실례로 다음의 정의가 그러하다.

```
float sum = .1+.1+.1+.1+.1+.1+.1+.1+.1+.1
```

다음의 식은 수학적으로는 true이지만 프로그람적으로는 true라고 말할수 없다. 이런 경우에는 같은가 같지 않은가를 직접 검사하는것보다 2개의 값이 충분히 서로 류사한가를 검사하는것이 나을것이다. 즉 최대의 오유오차를 규정하고 그 허용오차보다 비교대상들의 차의 절대값이 작다는것을 증명하는 방법으로 검사를 진행한다. 실례로 최대허용오차가 다음과 같은 경우

```
const float Delta = 0.0001;
```

다음의 식으로 2개의 값이 충분히 근사한가를 검열할수 있다. math서고(math서고는 5장에서 취급한다)함수 fabs()를 리용하였는데 이 함수는 그의 류점수파라메터의 절대값을 귀환한다.

```
fabs(sum -1.0) <= Delta
```

문제

1. 다음과 같이 객체를 정의한다.

```
bool q1 = true;
bool q2 = true;
bool q3 = false;
```

이때 다음의 식의 결과는 무엇인가?

```
(q1 && q2) || q3
```

2. 다음과 같이 객체를 정의한다.

```
bool q1 = true;
bool q2 = true;
bool q3 = false;
```

이때 다음의 식의 결과는 무엇인가?

```
(!(q1 && q2)) || q3
```

3. 다음의 정의가 있다.

```
bool  q3 = false;
```

이때 다음의 식의 결과는 무엇인가?

```
(!(!(!q3)))
```

4. 다음의 정의가 있다.

```
bool q1 = true;
bool q2 = true;
bool q3 = false;
```

이때 다음의 식의 결과는 무엇인가?

```
(q3 || q1) && q2
```

5. 다음의 정의가 있다.

```
bool q1 = true;
bool q2 = true;
bool q3 = false;
```
이때 다음의 식의 결과는 무엇인가?
```
(!(q3 && q1) && q2
```
6. 다음의 정의가 있다.
```
bool q1 = true;
bool q2 = true;
bool q3 = false;
```
이때 다음의 식의 값은 무엇인가?
```
(!(!q3 && q1)) || q2
```
7. 다음의 정의가 있다.
```
bool q1 = false;
bool q2 = true;
bool q3 = true;
```
이때 다음의 식의 결과는 무엇인가?
```
(!(!q3 && q1)) || q2
```
8. 다음의 정의가 있다.
```
bool q1 = false;
bool q2 = true;
bool q3 = false;
```
이때 다음의 식의 결과는 무엇인가?
```
(!(!q3 && q1) || q2
```
9. 다음의 정의가 있다.
```
int i = 5;
int j = 10;
int k = 30;
```
이때 다음의 식의 결과는 무엇인가?
```
( i < j ) && ( k > 10)
```
10. 다음의 정의가 있다.
```
int i = 5;
int j = 10;
int k = 30;
```
이때 다음의 식의 값은 무엇인가?
```
( i < j ) && ( k > j)
```
11. 다음의 정의가 있다.
```
int i = 5;
int j = 10;
int k = 30;
```

이때 다음의 식의 값은 무엇인가?

(i <= 10) && (k == 30)

12. 다음의 정의가 있다.

```
int i = -3;
int j = 10;
int k = 30;
```

이때 다음의 식의 값은 무엇인가?

(i != j) && ((i < 5) || (j < 10) || (k >= 30))

13. 다음의 정의가 있다.

```
int i = -3;
int j = 10;
int k = 30;
```

이때 다음 식의 값은 무엇인가?

((i < 5) || (j > 90)

14 . 다음의 정의가 있다.

```
int i = -3;
int j = 10;
int k = 30;
```

이때 다음의 식의 값은 무엇인가?

(i != j) && ((i < 5) && (j < 10) || (k >= 30))

15. 다음의 정의가 있다.

```
int i = 3;
int j = 10;
int k = 30;
```

이때 다음의 식의 값은 무엇인가?

(i != j) && (((i < 5) || (j < 10)) && (k >=30))

16. 다음의 정의가 있다.

```
int i = 3;
int j = 10;
int k = 30;
bool b1 = false;
```

이때 다음의 식의 값은 무엇인가?

(i != j) && (((i < 5) || (j < 10)) && b1)

17. 다음의 정의가 있다.

```
int i = 3;
int j = 13;
int k = 30;
bool b1 = true;
```

이때 다음의 식의 값은 무엇인가?

(i != j) && (((i < 5) || (j < 10)) && b1)

4.3 if명령문에 의한 조건실행

if명령문에는 두가지 형식이 있다. 두가지중 간단한것은 다음과 같은 문법을 가진다.

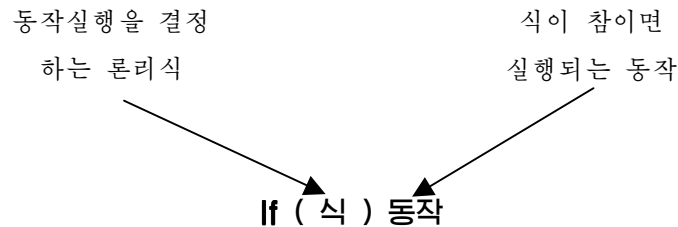



그림 4-1. if명령문

명령문이 프로그람안에서 실행될 때 if예약어다음에 있는 괄호안의 식이 평가된다. 《식》이 《참》이면 《동작》(가장 간단한 형식에서 《동작》은 한행으로 명령문을 구성할수 있다.)이 실행된다. 《참》이 아니면 《동작》이 실행되지 않는다. 론리식이 어떻든지간에 프로그람은 다음명령문을 계속 실행한다.

if명령문의 실행처리과정은 론리적과정이다. 그 의미를 그림 4-1에 보여 주었다. 이 그림을 흐름도라고 하는데 프로그람실행흐름을 지적한다.

프로그람 4-1은 if명령문을 리용하여 입력값의 절대값을 구하는 프로그람이다. 입력값은 Value객체에 기억된다. 식 (Value<0)이 참이면 Value는 부수라는것을 의미하며 따라서 보충적인 값을 값주기하여 같은 크기를 가진 정수로 Value를 바꾼다. 식이 거짓이라면 Value는 정수이므로 아무런 동작도 수행할 필요가 없다. 프로그람 4-1에서 그리고 다른 실례들에서 기본적으로 설명문을 달아 주었다. 설명문을 리용하면 새로운 개념을 더 잘 알수 있다. 실지 이러한 설명들은 실행되지 않는다.

프로그람 4-1의 흐름도를 그림 4-2에 주었다.

```
//프로그람 4-1: 입력값을 절대값으로 변환한다.
#include <iostream>
#include <string>
using namespace std;
int main( ) {
    cout << "Please enter a number:" ;
    int Value;
    cin >> Value;
    if (Value < 0) //값이 0보다 작은가를 평가
        Value = -Value;  //작다면 부호를 변화시킨다.
    cout <<Value << " is positive" << end1;
    return 0;
}
```

프로그람 4-1. if명령문실례

아래의 코드토막에서 if명령문을 리용하여 입력값이 짝수인가를 결정한다. 프로그람 4-1에서처럼 입력값은 Value객체에 기억된다.

(Value % 2)는 Value를 2로 나눈 나머지값이다. 그 값이 0이 아니면 입력값은 짝수가 아니다.

```
cout <<"Please enter a number:" ;
int Value;
cin >> Value;
cout << Value <<"is";
if ((Value %2) != 0) //값이 홀수인가?
    cout << "not";  //값은 홀수이다.
    cout << "even" <<end1;
```

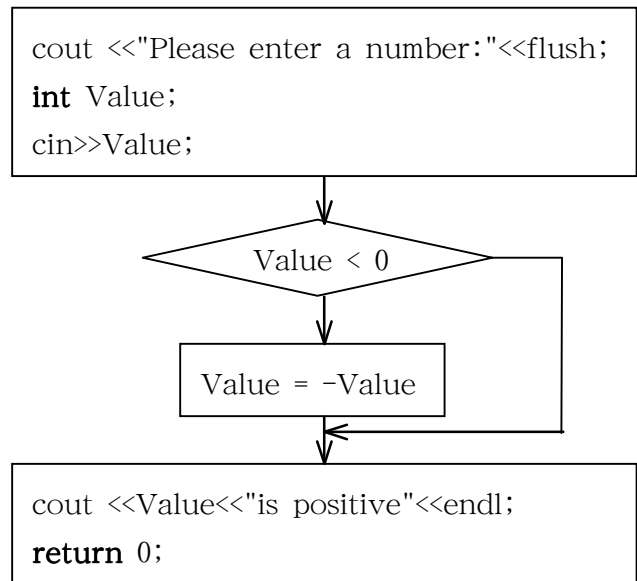
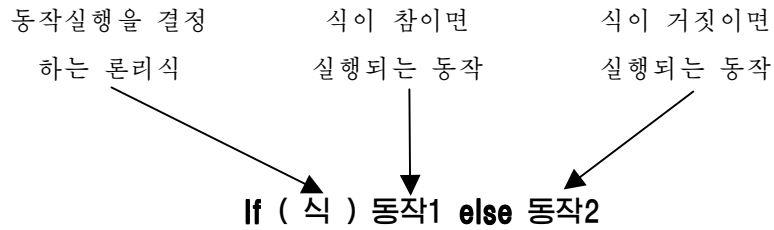
그림 4-2. 프로그람 4-1의 흐름도

식의 값에 따라 여러개의 명령문이 실행되여야 할 경우가 있다. 이러한 명령문블로크가 실행되도록 하려면 블로크안에 있는 명령문들을 왼쪽과 오른쪽 대괄호로 막아 주어야 한다. 블로크내의 개별적명령문들은 반두점으로 구분한다. 반두점이 반드시 필요하다. 다음의 코드토막은 이런 형식의 if명령문을 리용하여 2개의 입력값 Value 1 과 Value2를 입력하고 순서대로 출력하는 실례이다.

```
cout << "Please enter two numbers:";
int Value1;
int Value2;
cin >> Value1 >> Value2;
if (Value1 > Value2) {
    //Value1이 더큰가?
    //Value1이 더 크면 교체해야 한다.
    int RememberValue1 = Value1;
    Value1 = Value2;
    Value2 = RememberValue1;
}
cout << "입력수는 순서대로 놓이였다:"
  << Value1 << " " <<Value2 << end1;
```

이 코드토막에서는 2개의 수를 입력한후에 그것들을 비교한다. Value1이 Value2보다 크다면 두 값을 교환한다. RememberValue1객체는 교환용으로 리용된다. if명령문이 완성되면 Value1과 Value2가 표시된다.

4.3.1 if-else명령문

if명령의 두번째 형식은 론리식의 값에 기초하여 서로 다른 동작이 수행되여야 하는 프로그람작성에 리용한다. 이 형식은 다음과 같다.

여기서 동작1과 동작2는 반두점을 가진 한개 명령문이거나 대괄호로 막은 명령문블로크이다. 이런 형식의 **if**명령문이 실행될 때 식이 true이면 동작1이 수행되며 아니면 동작2가 실행된다. 그림 4-3에 있는 흐름도는 **if-else**의 의미를 보여 주었다. S와 T가 RectangleShape형객체로 초기화된다고 하면 다음의 코드토막은 정확히 이 두개의 객체가 같은 색갈을 가지는가를 식별한다.

```
if(S.GetColor()==T.GetColor())      //같은 색인가?
    cout <<"4각형은 같은 색이다.";
    else                            //색이 다르다.
        cout <<"4각형은 색이 다르다.";
    cout << end1;
```

그림 4-3. if-else명령문의 흐름도

색이 같으면 같기연산자는 true을 되돌리며 명령문

```
cout <<"4각형은 같은 색이다.";
```

가 실행되여 색이 같다는 통보를 출력한다.

색갈이 다르면 같기연산자는 false를 되돌리며 명령문

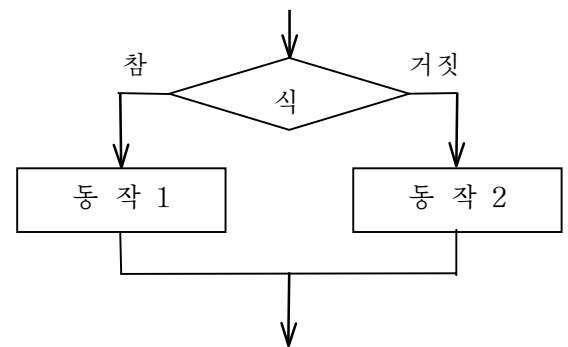
```
cout <<"4각형은 색이 다르다.";
```

가 실행된다. 그다음 명령문

```
cout << end1;
```

이 실행된다. 다음의 코드토막은 2개의 값을 입력하여 큰 값을 현시한다.

```
cout << "2개의 수를 입력하시오:";
int Value1;
int Value2;
cin >> Value1 >> Value2;
int larger;
if (Value1 < Value2) //Value2가 더 큰가?
    larger = Value2; //Value2가 더 크다.
  else //(Value1 >= Value2)
    larger = Value1; // Value1가 더 크다.
cout <<"The larger of" << Value1 << "and"
  << Value2 << "is" << larger << end1;
```

Value1과 Value2를 입력한후 코드토막은 (Value1<Value2)를 평가한다.

```
if(Value1 < Value2) // Value2가 더 큰가?
    larger = Value2;  //Value2가 더 크다.
else //(Value1 >= Value2)
    larger = Value1;  //Value1이 더 크다.
```

식이 참이면 입력한 두 수중에서 Value2가 더 크다는것이며 그것은 larger객체를 설정하기 위해 리용된다. 이 값주기명령문다음의 반두점은 true일 때 동작이 완성된다는것을 지적한다.

주의

들여쓰기와 대괄호의 위치

이 장에 있는 프로그람을 보면 if명령문에 결합된 《동작》명령문이 들여쓰기되여 있다는것을 알수 있다. 들여쓰기는 《동작》명령문이 그의 앞에 있는 if명령문식에 종속되여 있다는것을 보여준다.

들여쓰기는 프로그람을 쉽게 분석할수 있게 한다. 거의 모든 프로그람작성자들은 보통 3∼4개문자 들여쓰기한다.

조종명령문과 결합된 《동작》에도 같은 들여쓰기형식이 리용된다. 대괄호의 위치는 경험에 따라 설정할수 있다. 대괄호의 위치는 다음과 같다. 열기대괄호는 if명령문의 첫행에 포함시킨다. 닫기괄호는 다음명령이 if명령문과 결합되여 있지 않다는것을 시각적으로 보여 주도록 마지막명령문과 같은 행에 배치한다. 다음의 코드는 들여쓰기를 하지 않은 프로그람이다.

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    cout << "Please enter two numbers:";
    int Value1;
    int Value2;
    cin >> Value1 >>Value2;
    if (Value1 > Value2) {
        int RememberValue2  = Value1;
        Value1 = Value2;
        Value2 = RememberValuer;
    }
    cout << "The input numbers in sorted order:" <<
    Value1 << " " << Value2 << end1;
    return 0;
}
```

식 (Value1 < Value2)가 false이면 Value1은 Value2보다 크다는것을 의미한다. 따라서 else부분에서 Value1은 larger객체설정에 리용된다.

if-else가 완료될 때 larger가 적당히 설정되며 필요할 때 리용할수 있다. 프로그람 4-2는 3개의 값을 입력하고 그것들중에서 가장 작은 값을 선택하는 프로그람이다. 이 프로그람은 if-else명령문안에 또

다른 if-else명령문을 포함하고 있다.

3개의 입력값 Value1, Value2, Value3을 입력한 다음 Value1과 Value2를 비교한다. Value1이 Value2보다 작다면 Value1과 Value3을 비교한다. Value1이 Value3보다 작다면 Value1은 이 값들중에서 제일 작은 값이다.

```
//프로그람 4-2. 입력한 세값중에서 제일 작은 값을 결정하기
#include <iostream>
#include <string>
using namespace  std;
int main( ) {
    cout << "Please enter three numbers:"
    int Value1;
    int Value2;
    int Value3;
    cin >> Value1 >> Value2 >> Value3;
    int Smallest;
    if (Value1 <= Value2) {
        //Value1가 Value2보다 작다.
        if(Value1 <= Value3)
            //Value1이 Value3보다 작다.
            Smallest = Value1;
        else // Value3 < Value1
            // Value3이 Value1보다 작다면 그것은 Value2보다 작다.
            Smallest = Value3;
    }
    else { // Value2 < Value1
            // Value2는 Value1보다 작다.
            if (Value2 <= Value3)
                // Value2는 Value3보다 작다.
                Smallest = Value2;
            else
                Smallest = Value3;
    }
    cout << "The Smallest of"
      << Value1 << ", " << Value2 << ", and"
      <<Value3 << "is" << Smallest << end1;
    return 0;
}
```

프로그람 4-2. if명령문속에 또 다른 if명령문이 있는 경우의 프로그람

Value1이 Value3보다 작다면 Value1은 가장 작은 값으로 된다. Value1과 Value2의 초기비교에서 Value1이 Value2보다 크다면 Value2와 Value3을 비교하여 제일 작은 값이 Value1로 된다. 프로그람에서는 이것을 첫번째 if명령문과 결합된 else명령문안에서 결정한다. 앞서 언급한것처럼 들여쓰기형식은 프로그람을 빨리 리해할수 있게 할뿐 콤파일러에는 영향을 주지 않는다. 언어문법과 의미규칙은 프로그람이 어떻게 번역되는가를 정확히 결정한다. 이러한 규칙은 어느 if에 어느 else가 결합되는가를 결정하는데 애매한 점이 없도록 해야 한다. 이 규칙은 아주 간단하다. else앞에는 한개 명령문이나 명령문모임이 있어야 한다. 그리고 그앞에는 if가 있어야 한다.

else는 이 if와 결합된다. 그 실례는 다음과 같다.

```
if (p)
    if (q)
        cout << "A" << endl; //p는 참 , q는 참
    else // q가 아니다.
        cout << "B" << end1: //p는 참, q는 거짓
```

여기서 else와 그의 명령문은 q를 평가하는 if와 결합된다. 문자렬 "B"를 표시하자면 p가 true, q가 false이여야 한다.

다음의 코드토막에서는 대괄호가 내부if명령문을 둘러 막고 있다.

```
if (p) {
    if (q)
        cout <<"A" << end1; //p는 참, q는 참
    }
else
    cout << "B" <<  end1; //p는 거짓, q는 모른다.
```

이 괄호로 하여 else는 첫번째 if명령문과 결합된다. 따라서 문자렬 "B"를 표시하자면 p가 false이여야 하며 q값은 임의의 값이여도 된다.

4.3.2 3개 수의 배렬

때때로 여러개의 식중 어느것이 true인가 검사하고 적당한 동작을 실행하여야 하는 경우가 있다. 실례로 프로그람 4-3은 3개의 수를 입력하고 순서대로 수를 표시하는 프로그람이다. 즉 작아 지지 않는 순서로 배렬한다. 입력값이 같을수도 있으므로 커지는 차례로 배렬한다기보다 작아 지지 않는 순서로 배렬한다는 말이 적당하다.

3개의 수에 대하여 6개의 가능한 순서배렬이 있다.

- Value1 <= Value2 <= Value3
- Value1 <= Value3 <= Value2
- Value2 <= Value1 <= Value3
- Value2 <= Value3 <= Value1
- Value3 <= Value1 <= Value2
- Value3 <= Value2 <= Value1

프로그람은 먼저 Value1, Value2, Value3이 이미 배렬되였는가 검사한다. 배렬되여 있다면 그것을 output1, output2, output3에 그대로 복사한다. 즉 이 3개의 output객체는 정확히 배렬된것으로 된다. 검사식은

(Value1 <= Value2 <= Value3)

이 아니라

(Value1 <= Value2) &&(Value2 <= Value3)

이다. 웃식을 보면 수학적으로는 옳지만 프로그람작성에서는 잘못으로 된다. 연산자의 우선권으로 하여 그 식은

(Value1 <= Value2) <= Value3

과 같은데 이것은 론리값과 Value3을 비교하는것으로 된다.

```
//세수의 배렬
#include <iostream>
#include <string>
using namespace std;
int main() {
//입력을 얻고 출력을 정의한다.
    cout<<"Please enter three number:";
    int Value1;
    int Value2;
    int Value3;
    cin >>Value1 >> Value2 >> Value3;
    int output1;
    int output2;
    int output3;
    if ((Value1 <= Value2) && (Value2 <= Value3)) {
        output1 = Value1;
        output2 = Value2;
        output3 = Value3;
    }
    else if ((Value1 <= Value3) && (Value3 <= Value2)){
        output1 = Value1;
        output2 = Value2;
        output3 = Value3;
    }
    else if ((Value2 <= Value1 ) && (Value1 <= Value3)) {
        output1 = Value1;
```

```
        output2 = Value2;
        output3 = Value3;
    }
    else if ((Value2 <= Value3)  && (Value3 <= Value2)) {
        output1 = Value2;
        output2 = Value3;
        output3 = Value1;
    }
    else if ((Value3 <= Value1) && (Value1 <= Value2)) {
        output1 = Value3;
        output2 = Value1;
        output3 = Value2;
    }
    else { // (Value3 <= Value2 ) && (Value2 <= Value1)
        output1 = Value3;
        output2 = Value2;
        output3 = Value3;
    }
    cout << Value1 << ""<< Value2 <<""<< Value3
      <<" in sorted order is"<< output1 << output2
      <<"" <<output3 << end1;
    return 0 ;
}
```

프로그람 4-3. if-else-if구조에 대한 설명

즉 이 식에서는 어느것이 필요하며 어느것이 리용되지 말아야 하는가를 알수 없다. 3개 입력값이 순서대로 배렬되지 않으면 Value1이 가장 작고 Value3이 중간값, Value2가 가장 큰 값인가를 결정한다. 이 검사가 true이면 Value1은 output1에, Value2는 output3에, Value3은 output2에 복사한다.

만일 이 검사가 false이면 다른 경우를 검사한다. 검사처리는 5개의 경우에 대하여 조건이 만족될때까지 계속 분석한다. 프로그람 4-3에서 else if는 새로운 명령문이 아니다. 대신 그것들은 else를 재배치한것이며 그것들과 결합된 《동작》명령문들이다. 프로그람 4-3은 여러가지 식을 검사하고 true로 평가되는 첫번째 식과 관련된 《동작》을 수행한다.

문 제

18. 다음의 코드토막을 분석하시오.

```
int i = 5;
int j = 7;
int k = 6;
if ((i < j)&&( k < 5))
    cout << "Yes" <<endl;
```

```
else
     cout << "No" <<endl;
```

이때 출력은 무엇인가?

19. 다음의 코드토막을 분석하시오.

```
int i = 10;
int j = 7;
int k = 4;
if ((i < j) && ( k < 5))
     cout << "Yes" <<endl;
else
     cout << "No" <<endl;
```

이때 출력은 무엇인가?

경험

효과적인 조건식을 작성

조종구조에서는 조건식에 따라 수행될 동작이 규정된다. 조건식을 알기 쉽게 작성하는것은 오유가 없고 유지가 쉬운 코드를 작성하는 열쇠이다. 조건식은 될수록 간단히 작성하여야 한다.

첫째로 부정연산을 될수록 없애는것이다. 실례로

```
if (!(command == 'u' || command == 'U'))
     return;
```

은 드 모르간의 법칙을 리용하여 다음과 같이 간단히 할수 있다.

```
if (command != 'u' && command != 'U'))
     return;
```

마찬가지로 if명령문

```
if ( !InputOk) {
     //
     ...
}
else {
   //
   ...
}
```

은 부정연산자를 없애고 if-then-else 부분을 교체한다면 더욱 명백해 질것이다.

```
if(InputOk) {
     //
      ...
}
else {
     //
    ...
}
```

20. 다음의 코드토막을 분석하시오.

```
int i = 10;
```

```
int j = 7;
int k = 4;
if ((i >= j) || (5 < j))
    cout << "Yes" << end1;
else
    cout << "No"  << end1;
```

이때 출력은 무엇인가?

21. 옹근수 j가 25보다 작고 10보다 크면 cin흐름으로부터 옹근수형객체 Score에 옹근수값을 입력하는 C++코드를 작성하시오.
22. j가 5보다 작으면 k를 10으로 설정하고 아니면 k를 설정하지 않는 C++코드를 작성하시오.
23. 옹근수형객체 m이 5보다 작으면서 j가 0보다 작으면 옹근수 k를 3으로 설정하고 아니면 k를 설정하지 않는 C++코드를 작성하시오.
24. 옹근수형객체 m이 5보다 작으면서 j가 0보다 작으면 옹근수 k를 3으로 설정하고 아니면 k를 설정하지 않는 C++코드를 작성하시오.
25. m이 짝수이면서 3으로 나누어 지면 k를 34로 설정하고 아니면 k를 설정하지 않는 C++코드를 작성하시오.
26. 다음의 코드토막을 분석하시오.

```
int i = 10;
int j = 7;
int k = 4;
 if ((i >= 11) && (j != 7))
    cout << "Yes" << end1;
 else
    cout << "Maybe"  << end1;
```

이때 출력은 무엇인가?

27. 다음의 코드토막을 분석하시오.

```
bool A;
bool B;
bool C;
bool D;
if ( A && B)
    if ( !C && !D)
        cout << "1" << end1;
    else if (!D)
        cout << "2" << end1;
    else
        cout << "3" <<end1;
```

```
        else if( C == D)
            cout << "4" << end1;
        else if( C )
            cout << "5" << end1;
        else
            cout <<"6" <<end1;
```

표준출력지령으로 3을 표시하도록 A, B, C, D의 값을 주시오.

28. 세금을 계산하는 프로그람을 작성하시오. 가격이 30,000$~50,000$이면 세금비률이 4%이다. 가격이 50000$~70000$이면 세금비률이 4.5%이다. 70,000$이상이면 5%이다.

4.4 switch명령문에 의한 조건실행

쏘프트웨어기사는 때때로 식의 값에 따라 동작이 진행되는 프로그람을 작성하여야 할 경우가 있다. if-else-if구조를 리용하면 개별적인 비교를 하여 식과 값이 같으면 적당한 동작을 실행한다.

실례로 현재문자 command가 화면상에 있는 객체를 움직이는 유효한 지령문자('u' 'd' 'l' 'r' 상, 하, 좌, 우)인가를 검사한다고 하면 if명령문을 리용하여 아래와 같이 코드를 작성할수 있다.

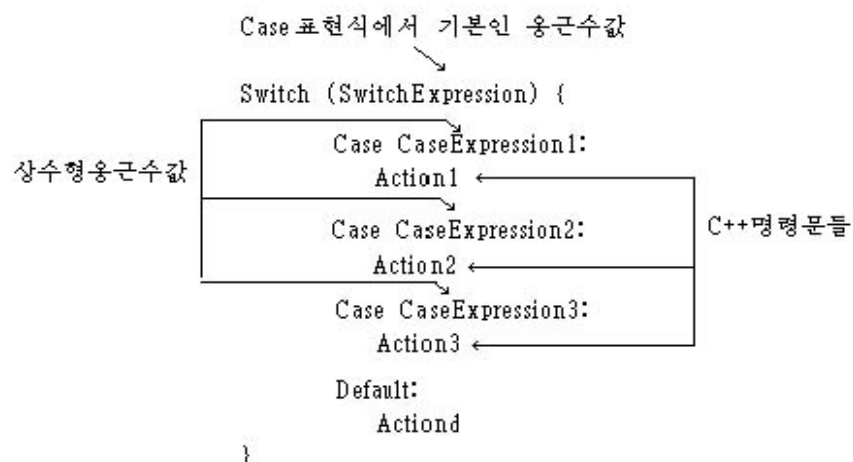
```
if (command == 'u')
    cout << "Move Up command received" <<end1;
else if (command == 'd')
    cout << "Move Down command received" << end1;
else if (command == 'I')
    cout << "Move Left command received" << end1;
else if (command == 'r')
    cout << "Move right command received" <<end1;
else
    cout << "Invalid command received" << end1;
```

이런 경우가 자주 생기기때문에 C++언어에 switch명령문이 있다. 이 명령문을 리용하면 조종문자의 검사를 보다 간결하고 보기 좋게 할수 있다.

```
switch (command) {
    case 'u':
        cout <<"Move Up command received" <<end1;
        break;
   case 'd':
        cout <<"Move Down command received" <<end1;
        break;
   case 'l':
        cout <<"Move Left command received" <<end1;
        break;
```

```
        case 'r':
            cout <<"Move Right command received" <<endl;
            break;
    }
```

switch명령문의 형식은 아래와 같다. case와 결합된 동작은 한개 명령문일수도 있고 명령문블로크일수도 있다. 동작이 명령문블로크일 때 대괄호가 전혀 필요없다. switch명령문이 실행될 때 조건식이 평가된다. 그 값이 조건과 같으면 조종지령이 조건식과 결합된 <동작 i >로 넘어 간다. 조건식이 다 맞지 않는 경우에는 기정동작 d가 실행된다.

웃실례에서 default부문에서 오유통보를 내보낸다. 만일 default가 없는 경우 switch다음의 명령문이 실행된다.

현실에서는 거의 모든 switch명령문에 default동작을 규정하여 기대되지 않은 조건을 검사한다. 선택된 동작을 수행한후 실행은 동작과 관련된 명령문에 따라 진행된다. 보통 동작의 마지막에 break명령문을 준다. break명령문은 switch명령문이 과제를 완성하였으며 조종흐름이 switch의 다음명령문에로 넘어 가야 한다는것을 지적한다. break명령문이 제공되지 않으면 조종이 다른 명령문에로 넘어 간다. 다음의 코드에서 i값이 3이면 "Hello, world"문자렬이 3번 표시된다. i값이 2이면 2번 표시된다.

```
    switch (i) {
        case 3;
                cout << "Hello, world" <<endl;
        case 2;
                cout <<"Hello, world" <<endl;
        default:
                cout << "Hello, world" <<endl;
    }
```

여기서는 조종이 다음명령문으로 넘어 가므로 모든 경우에 대하여 같은 개수의 문자렬이 출력된다.

i값이 2도 3도 아니면 기정동작이 수행되며 문자렬은 한번만 표시된다. 그림 4-4에 이 코드의 조종흐름을 보여 주는 흐름도를 주었다.

보통 요구하는 효과를 보기 위하여 switch명령문의 조건무시동작의 리용을 피하여야 한다. 동작은

break명령문으로 끝나야 한다. 그러나 조건무시동작을 필요로 하는 경우가 있다.

조건식의 서로 다른 조건값에 대하여 같은 동작을 진행하려면 실례로 대소문자의 구별이 없이 조종문자를 입력한다고 할 때 조건무시동작을 리용할수 있다.

실례로 대소문자 'u'를 처리하기 위한 case명령문은

```
case 'u':
case 'U':
     cout << "Move up command received" <<endl;
     break;
```

이다.

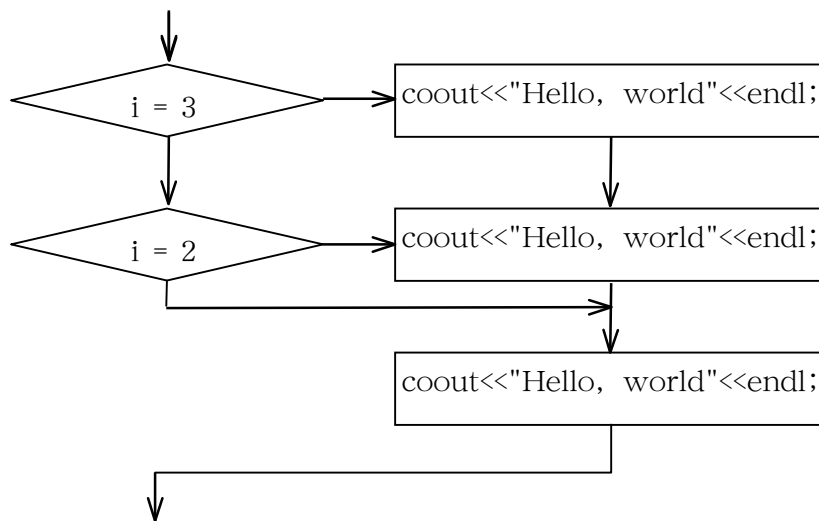



그림 4-4. break명령이 없는 switch명령의 흐름도

case 'u'인 경우에는 동작이 비여 있으므로 조종이 case'U'로 넘어 가 그의 동작이 실행된다.

다음의 코드는 이 수법을 리용하여 대소문자를 구별함이 없이 입력한 조종문자에 따라 동작을 진행한다. 같은 동작을 수행하는 case명령문은 한행에 쓰는것이 보통이다.

```
switch(command) {
   case 'u': case 'U':
      cout << "Move Up command received" <<endl;
      break;
   case 'd': case 'D':
      cout << "Move Down command receivedif0<<endl;
      break;
   case 'l': case 'L':
      cout << "Move Left command receivedif0<<endl;
      break;
   case 'r': case 'R':
      cout << "Move right command received"<<endl;
```

```
        break;
    default:
        cout <<"Invalid command receivedif0<< end1;
}
```

조건식과 개별적인 조건들이 다 옹근수여야 하므로 switch명령문은 류점수객체에 기초한 동작을 결정하는데 리용할수 없다. 그러한 처리는 if-else-if명령문을 리용하여 진행한다.

경험

switch명령문의 효과적리용

switch명령문을 적당히 리용하면 프로그람을 보다 알기 쉽고 효과적으로 작성할수 있다. 여기에 switch명령문을 효과적으로 리용하기 위한 방법을 소개한다.

우선 동작코드가 짧아야 한다. 이렇게 하면 switch명령문의 구조를 명백히 할수 있다. 여러 페지에 작성한 동작코드는 구조가 명백치 않다. 다음 암시적인 경우를 제공하여야 한다. 이렇게 하여야 오유가 검출되지 않는 현상을 막을수 있다. 마지막으로 조건충돌을 피하여야 한다. 이렇게 하지 않으면 코드를 리해하기도 유지하기도 어렵다. 조건충돌이 적합한 경우는 서로 다른 조건에 대하여 같은 동작을 수행할 때이다. 이때에는 이 조건들을 일정한 규칙으로 배렬하여야 한다. 즉 자모순에 따라 배렬하든가 빈도수에 따라 배렬하여야 한다.

4.5 수값계산

한개의 연산자에 해당한 계산을 진행하고 현시하는 간단한 수산기프로그람을 작성하여 보자. 여기서 제기되는 문제는 다음과 같다.

산수연산자에 의해 분리된 한쌍의 연산수를 입력하여야 한다. 입력한 연산수의 의미가 옳으면 계산을 진행하여 표준출력지령으로 결과를 현시하여야 한다. 만일 연산자를 정의하지 않았거나 연산수의 의미가 옳지 않으면 오유통보가 표준출력흐름에 현시되여야 한다.

실례로 입력값이 15*20이라면 프로그람은 다음과 같이 현시되여야 한다.

    15*20 = 300

만일 연산자가 무효하다면 즉 24～25이라면 프로그람은 다음과 같은 결과를 현시한다.

    ～ is unrecognized operation

아래와 같이 나누기연산자의 분모값을 0으로 입력하면

    23/0

프로그람은 다음과 같은 통보를 현시한다.

    23/0 cannot be computed: denominator is 0

문제를 풀기 위한 알고리듬을 간단히 세 단계로 서술할수 있다.

1단계. 입력재촉문에서 계산하기 위한 값을 입력한다. 연산수값은 옹근수이다. 또한 문자도 될수 있다.

2단계. 입력값을 검사하고 해당한 계산을 진행한다.
- 단계2.1 연산자의 형을 결정한다.
- 단계2.2 연산수가 적합한가를 결정한다.

단계2.3 입력연산자와 연산수에 해당한 값을 계산한다.

3단계. 결과를 현시한다.

배렬이 정확하고 매 단계에서 정확히 진행된다면 이러한 알고리듬은 쉽게 문제를 해결할것이다. 이 알고리듬을 실현한 프로그람이 프로그람 4-4이다. 1단계와 2단계는 C++서고함수를 리용하여 간단히 실현할수 있다. 2단계는 **switch**명령문으로 실현한다. 이 단계에서 연산자의 결과값은 Result객체에 보관된다.

switch명령문은 분리된 **case**명령문으로서 4개의 산수연산자를 취급한다. 즉 더하기, 덜기, 곱하기, 나누기, 이밖에 무효한 연산자인 경우가 있다. 더하기나 덜기 그리고 곱하기의 결과는 즉시에 결정할수 있다. 나누기인 경우에는 입력한 오른쪽연산수가 0이 아닌가를 검사하여야 한다. 이 값이 0이 아니라면 나누기연산이 진행된다.

```
//간단한 산수식계산
#include <iostream>
#include <string>
using namespace std;
int main() {
// 연산수를 입력한다.
cout << "Please enter a simple expression"
  << "(number operator number):";
    int LeftOperand;
    int RightOperand;
    char Operator;
    cin >> LeftOperand >> Operator >> RightOperand;
    //연산자를 확인하고 해당계산을 진행한다.
    int Result;
    switch(Operator) {
      case '+';
            Result = LeftOperand + RightOperand;
            break;
      case '-';
            Result = LeftOperand - Right Operand;
            break;
      case '*';
            Result = LeftOperand * Right Operand;
            break;
      case '/';
            if (RightOperand != 0)
                Result = LeftOperand / RightOperand;/RightOperand;
            else {
                cout << LeftOperand << "/"
```

```
                        << RightOperand <<"cannot be computed:"
                        << " denominator is 0." << end1;
                    return 1;
                }
                break;
        default:
                cout << Operator
                  << " is unrecognized operation." << end1;
                return 1;
        }
        //결과표시
        cout << LeftOperand << " " << Operator << " "
          << RightOperand << " equals " << Result << end1;
        return 0;
}
```

프로그람 4-4. 입력한 산수값을 계산하는 프로그람

만일 이 값이 0이라면 오유를 내보내고 프로그람은 0아닌 값을 돌려 주게 된다. 프로그람에서 돌려주는 값이 0이라는것은 계산이 성과적으로 진행되였다는것을 의미한다. 되돌이값이 0이 아니면 요구한 동작이 정확히 진행되지 않았다는것을 의미한다. switch명령문의 흐름도를 그림 4-5에 보여 주었다.

4.6 날자계산

다음으로 사용자로부터 날자를 입력받고 그 날자의 유효성을 검사하는 프로그람을 개발하여 보자. 프로그람은 년간 날자(실례로 2월 12일은 그 해의 43번째 날이다)를 계산하는 련습문제를 통하여 더욱 심화시키자.

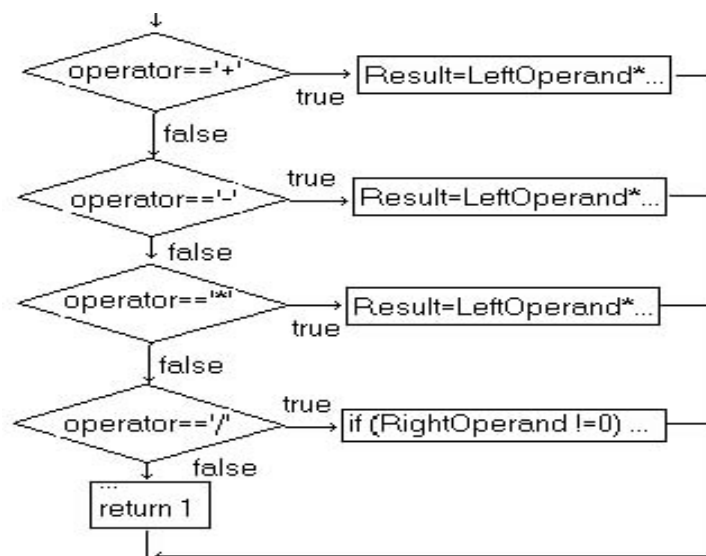
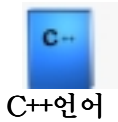



그림 4-5. break문을 리용한 스위치문흐름도

C++언어

? :연산자

? :연산자는 흔히 조건문에서 리용될수 있다. 연산자의 리용형식은 다음과 같다.

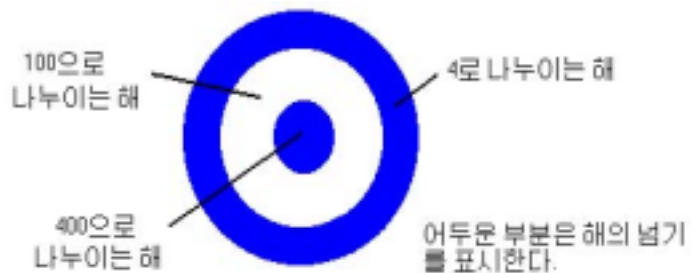
TestExpression ? Expression1 : Expression2;

이것을 실행하면 먼저 TestExpression이 평가된다. 만일 TestExpression이 **true**이면 연산결과는 Expression1이며 그렇지 않으면 결과가 Expression2이다. 2개의 결과식은 두점(:)에 의하여 분리된다. 이 연산자는 2개의 수중 작은 값을 찾는것과 같은 프로그람에서 리용된다. 아래의 실례를 보도록 하시오.

```
int  input1:
int  input2:
cin >> input1 >> input2;
int  Min = input1 <= input2 ? input1 : input2;
```

문제에 대한 알고리듬을 만들기전에 먼저 그 해가 윤년인가를 결정하기 위한 규칙을 보기로 하자. 윤년이 되자면 그 해가 4로 완전히 나누어 져야 한다. 그러나 완전히 4로 나누어 지는 해들 모두가 윤년인것은 아니다. 마지막 2개의 수자들이 령인 해들은 "세기"년들이다. 례를 들어 1800, 1900, 2000은 "세기"년들이다. 4로 나누어 지지 않는 세기년들은 항상 윤년이 아니다. 그러나 만일 세기년들이 400으로 나누어 지기만 한다면 윤년이다. 1600년과 2000년은 윤년이다. 1700, 1800, 1900년들은 윤년이 아니다.

그림 4-6. 벤도표

이 규칙들을 그림 4-6에 주어 진 벤(Venn)도표로 주었다.

윤년을 결정하기 위한 규칙을 리용하여 아래와 같은 알고리듬을 설계한다.

단계1. 년, 월, 일을 입력한다.

단계2. 년이 윤년인가 아닌가를 결정한다.

단계2.1 만일 년이 4로 나누어 지지 않으면 그 해는 윤년이 아니다.

단계2.2 만일 년이 400으로 나누어 진다면 그 해는 윤년이다.

단계2.3 만일 년이 세기년이고 400으로 나누어 지지 않으면 그 해는 윤년이 아니다.

단계2.4 우의 조건이 맞으면 그 해는 윤년이다.

단계3. 날자를 확인한다.

단계3.1 만일 월이 무효하면 오유통보문을 현시하고 완료한다.

단계3.2 월에서 날자가 무효이면 오유통보문을 현시하고 완료한다.

단계3.3 월과 일이 유효하면 날자가 유효하다는 통보를 표시한다.

단계1을 해석하는것은 간단하다. 그리고 그것을 실현하는 코드는 프로그람 4-5에 있다. 단계2를 해석하는것은 입력년도가 4가지 경우중 어느것인가를 결정하는것으로써 실현할수 있다. 이 단계에서 중요한 부분은 주어 진 년에서 2월달이 몇개의 날자를 가지고 있는가를 결정하는것이다. 이 정보는 객체 DaysInFebruary에 있다. **if-else-if**구조를 리용하여 그 객체를 설정할수 있다.

```
int DaysInFebruary ;
if ((Year % 4 ) ! = 0 )
```

```
        DaysInFebruary = 28 ;
    else if  (( Year % 400) = = 0)
        DaysInFebruary = 29 ;
    else if ((Year % 100) = = 0 )
        DaysInFebruary = 28;
    else
       DaysInFebruary = 29 ;
```

앞선 코드토막에서 만일 검사식 ((Year % 4) != 0)이 true이면 그때 입력년도는 윤년이 아니며 2월은 28을 가진다. 대신에 ((Year % 4) != 0)이 false이면 윤년이다. 검사식 ((Year % 400) == 0)이 true이면 자동적으로 윤년으로 결정되며 따라서 2월은 29일이다. 대신 검사식((Year % 400) == 0)이 false이고 입력된 년이 100으로 나누어 지지 않아도 윤년일수 있다. 마지막 else-if검사는 이것이 true인가 아닌가를 결정한다.

단계3에서는 먼저 입력한 달이 몇일까지인가를 결정하고 월의 첫날과 마지막날사이에 입력한 날자가 놓이는가를 확인한다. 입력한 달의 날자수는 입력한 달이 어느 부류 즉 31일을 가진 달인가 30일을 가진 달인가 혹은 2월인가에 따라 계산될수 있다. if-else-if로도 할수 있지만 switch문이 더 좋을것이다.

```
    int  DaysInMonth;
    switch (Month){
        case January : case March: case May: case July:
        case August : case October: case December:
                DaysInMonth = 31 ;
                break;
        case April : case June: case September: case November:
                DaysinMonth = 30 ;
                break ;
        case  February :
                DaysInMonth =DaysInFebruary;
                break ;
        default :
                cout<<"Invalid month:"<<Month<<endl;
                return 1;
    }
```

여기서 월을 나타내는 상수들은 미리 정의되여야 한다. 이 상수들의 정의는 두가지 방법으로 진행할수 있다. 한가지 방법은 매 상수를 개별적으로 정의하는것이다.

```
    const  int     January = 1;
    const  int     February = 2;
    const  int     March = 3 ;
    const  int     April = 4;
    const  int     May = 5 ;
```

```
const  int     June = 6 ;
const  int     July = 7 ;
const  int     August = 8 ;
const  int     September = 9 ;
const  int     October = 10 ;
const  int     November = 11 ;
const  int     December = 12 ;
```

그러나 C++는 관련된 기호상수들의 집합을 정의하기 위하여 **enum**형을 제공하고 있다. 상수들의 집합은 하나의 자료형을 이룬다. **enum**정의에서 상수들은 값이 커지는 순서로 기입된다.

값들이 1부터 12까지인 MonthsOfYear형을 정의하기 위하여 다음의 **enum**문을 리용할수 있다.

```
enum  MonthsOfYear  { January =1, February=2, March=3,
April=4, May=5, June=6, July=7, August=8, September=9,
October=10, November=11, December=12 };
```

이러한 형들을 사용자정의형이라고 한다. **enum**은 보통 이런 경우에 리용한다. 그것은 **enum**이 프로그람작성자로 하여금 파생된 형의 객체를 정의하게 하기때문이다. 그러한 정의는 객체 그자체의 이름은 주지 못하더라도 코드를 읽어 보는 사람이 객체를 위한 가능한 값에 대하여 더 잘 알수 있게 한다. 실례로:

```
MonthsOfYear M ;
MonthsOfYear BirthdayMonth = April ;
MonthsOfYear Springbreak = March ;
```

에서 **enum**상수들은 **int**형객체가 아니며 이것은 **enum**형객체들에 대하여 **int**형산수연산자들을 리용할수 없다는것을 의미한다. 또한 **enum**상수에 대하여 인용괄호를 사용할수 없다는데 대하여 주의하여야 한다. 실례로 년의 네번째 달에 대하여 April을 사용하여야 하며 "April"은 사용하지 않는다. April은 식별자이지만 "April"은 문자렬이다.

입력한 달이 유효하면 **switch**문은 정확히 DaysInMonth객체를 할당한다. 무효한 달은 **switch**문의 **default**에 의하여 처리된다. 무효값이면 적당한 오유통보문이 표시되도록 하며 령 아닌 값(실례로)을 돌려 주는것으로써 프로그람을 완료한다.

DaysInMonth객체가 결정된 다음 **if**문은 Day가 유효한가를 검사하는데 리용할수 있다. Day가 1보다 작거나 DaysInMonthl을 초과하면 오유통지문이 표시되며 프로그람은 귀환값 1을 가지고 완료한다.

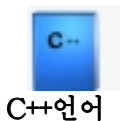
```
if((Day < 1?|| (Day > DaysInMonth)){
      cout << "Invalid day of Month :"<< Day<< endl ;
    return 1 ; }
```

C++언어

값을 주지 않은 상수의 정의

enum형 MonthOfYear는 상세한 값들이 필요하지만 프로그람작성자는 **enum**상수들에 실제 값들을 늘 주지 않아도 된다. C++에서는 상수에 자동적으로 값을 할당한다.

```
enum Music { Class : Cal,Country, Jazz, Popular, Soul, Rap, Rock };
```

RectangleShape객체들의 색을 서술하는데 리용되는 color형도 역시 자동적으로 정의된다.

enum color{ White, Red, Green, Blue, Yellow, Cyan, Magenta} ;

만일 프로그램작성자가 값을 제공하지 않는다면 목록에서 첫번째 상수는 0이며 두번째 상수는 1로 될것이다.

식 ((Day < 1) || (Day > DaysInMonth))이 **false**이면 입력한 날자는 유효하다. 식이 **true**이면 **return**문의 실행으로 프로그람이 완료되므로 **else**문이 있는 프로그람에서 나머지동작을 갈라볼 필요는 없다. 따라서 프로그람실행이 이 **if**문다음에 오는 출력명령문에 이르면 Day가 정확한 값을 가지게 된다는것을 알수 있다.

날자확인프로그람의 해석과 개발을 이것으로 끝낸다. 프로그람 4-5를 통하여 완전히 리해하기 바란다.

```
//Program 4-5 : Determine whether user date is valid
#include <iostream>
#include <string>
using namespace std;
int main( ) {
    enum MonthsOfYear { January =1, February =2, March =3,
    April =4, May =5, June=6,July =7, August = 8,September=9,
    October=10, November=11, December=12 };
    //Prompt and extract date
    cout << if please supply a date (mm dd yyyy):if0
    int Month;
    int Day;
    int Year;
    cin >> Month >> Day >> Year;
    // compute days in February
    int DaysInFebruary;
    if ((Year % 4) !=0)
        DaysInFebruary =28;
    else if  ((Year % 400)== 0)
        DaysInFebruary =29;
    else if (( Year %100) ==0)
        DaysInFebruary  =28;
    else
        DaysInFebruary = 29;
    // if month is valid, determine how many days it has
    int DaysInMonth;
    switch (Month){
      case January: case March: case May: case July:
      case August: case October: case December:
          DaysInMonth =31;
```

```
            break;
        case April: case June: case September:
        case November:
            DaysInMonth =30;
            break;
        case February:
            DayInMonth = DaysInFebruary;
            break;
              cout << doNovalid month:if0<Month<<endl;
            return 1;
    }
    // determine whether input day is valid
    if ((Day < 1)|| (Day>DaysInMonth)) {
        cout << do0nvalid day of month:do0<< Day <<endl;
        return 1;
    }
    // display result
    cout << Month << ":"<<Day<< ":"<<Year << "is a valid date" <<endl;
    return 0;
}
```

프로그램 4-5. 입력날자가 유효한것인가를 결정

문 제

29. 다음의 코드를 switch문으로 바꾸시오.

```
int i;
int n;
int k;
if (I == 3){
    n = 1 ; k = 5 ; }
 else  if ( i == 4 )
    n = 5 ;
 else if (i == 6)
    n = 6 ;
else    n = 0 ;
```

30. 다음의 코드를 switch문으로 바꾸시오.

```
int i ;
int  n ;
bool  TryAgain ;
```

```
if ( i == 3 || i == 5 ){
        ++n;
        TryAgain = false ; }
 else  if ( i = = 4 || i = = 10 )
         n = 5 ;
 else  if ( i = = 6 )
        n = 6 ;
 else{   n = 0;
        TryAgain = true ;
     }
```

31. 다음의 switch문을 분석하시오.

```
switch (i++) {
     case 1 : case  2: case 3:
         cout << " yes "<< endl ;
         break;
      case 5: case 6 :
         cout << "Maybe"<< endl ;
          break ;
      default:
          cout<< "Sometimes"<< endl;
}
```

i가 4이면 우의 코드의 출력은 무엇인가?

32. 다음의 switch문을 분석하시오.

```
switch (++i) {
       case 1  : case 2: case 3 :
            cout << "Yes"<< endl;
       case 5 : case 6 :
            cout << "No"<< endl ;
       case 10 :  case  11 :
            cout<< "Maybe"<< endl;
            break;
       default:
            cout<< "Sometimes"<<endl ;
}
```

i가 6이면 우의 코드의 출력은 무엇인가?

33. cin지령으로부터 한 문자를 얻는 프로그람을 실행하시오. 문자는 색을 가리킨다. 가능한 문자들은 'y'(노란색), 'r'(적색), 'b'(청색), 'g'(록색), 'm'(자홍색)이다. 입력이 'y'라면 프로그람은

2×3cm인 노란색4각형을 그려야 하며 입력이 ' r '이라면 프로그람은 3×3cm인 적색4각형을, 입력이 ' b '이라면 프로그람은 1×2cm인 청색4각형을, 입력이 'g'이라면 프로그람은 3×4cm인 록색4각형을, 그리고 입력이 'm'이라면 프로그람은 4×4cm인 자홍색4각형을 그려야 한다. 모든것들은 창문의 중심에 그려 져야 한다. 창문은 폭이 8cm이고 높이가 6cm여야 한다. 입력이 정확한 문자가 아니라면 문양이 그려 지지 말아야 하며 오유통지문이 cerr지령에 씌워져야 한다.

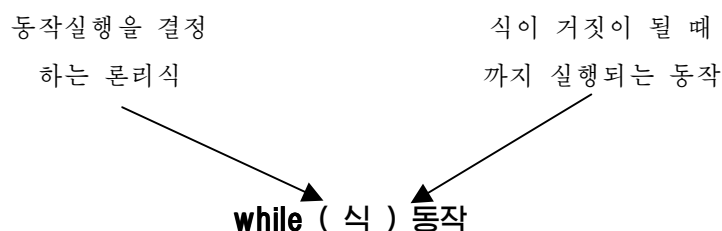
4.7 while명령문에 의한 순환

표준입력지령으로부터 얻는 다섯개 수자목록의 평균을 계산하려고 한다고 가정하자. 다음의 코드토막과 같이 쓸수 있다.

```
float  Value 1;
float  Value 2;
float  Value 3;
float  Value 4;
float  Value 5;
cin>> Value 1>> Value 2>> Value 3>> Value 4>> Value 5;
float Average = (Value1+Value2+Value3+Value4+Value5)/5;
```

이때 1000개 값을 가지는 목록의 평균을 계산할 필요가 있다고 가정하자. 간단히 앞의 코드를 수정하기는 너무 어렵다. 더 좋은 방법은 련속적으로 다음입력값을 얻고 지금까지 처리된 값들의 합에 그 값을 더하는 순환구성요소를 가진 코드토막을 쓰는것이다.

단계 1 : 실행합을 0으로 설정.
단계 2 : 지금까지 처리된 값의 수를 0으로 설정.
단계 3 : 지금까지 처리된 값의 수가 목록크기와 같다면 단계6에로 간다.
단계 4 : 다음값처리
　　단계4.1 : 다음값얻기
　　단계4.2 : 새로운 값을 값들의 실행합에 더하기
　　단계4.3 : 지금까지 처리한 값들의 수를 하나 증가
단계 5. 단계3을 반복.
단계 6. 평균을 계산하기 위하여 실행합을 목록크기로 나누기.

프로그람을 련속적으로 실행시키기 위한 간단한 방법은 while문을 통하여 이루어 진다. 구조는 다음의 형태를 가진다.

여기서 식은 론리식이며 동작은 하나의 명령문이거나 대괄호안에 넣어 진 명령문들의 모임이다. 동작을 흔히 **while**문의 본체라고 한다. **while**문이 실행되면 식이 먼저 평가된다. 식이 참이면 동작이 실행된다. 평가처리는 그다음 반복되며 식이 다시 참이면 동작이 실행된다. 이 처리를 순환이라고 하며 식이 거짓이 될 때까지 계속한다. 그 경우 프로그람은 다음명령문을 계속 실행한다. 이 처리를 그림 4-7의 흐름도로 주었다.

프로그람 4-6에서는 다섯개의 입력평균값을 간단히 줄수 있도록 **while**문을 리용한다. 프로그람은 처리하여야 하는 값의 수를 표현하기 위하여 상수 ListSize를 먼저 정의한다. 객체 ValueProcessed는 그다음 정의된다. 그것은 처리된 값의 수를 나타낸다. 값이 아직 처리되지 않았으므로 ValueProcessed는 0으로 초기화된다. 불변량은 프로그람동작에 대하여 프로그람작성자가 규정할수 있는 값이다. 프로그람 4-6을 보면 실행되는 동안 ValueProcessed의 값은 늘 주어 진 값으로 반영된다.

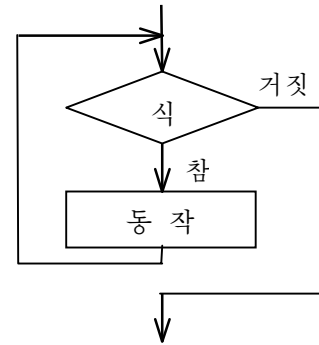



그림 4-7. While구조를 표시하는 흐름도

결과와 같이 ValueProcessed의 동작도 프로그람에서 불변량이다. 프로그람작성자의 요구가 객체동작에 대하여 불변이라는것을 확인하는것은 순환하는 프로그람에서 중요하다. 정확한 순환조작은 객체의 조건이 맞지 않을 때 중지될수 있다.

지금까지 처리된 입력값의 실행합은 ValueSum에 보관되는데 0으로 초기화 된다.

객체 ValueSum이 처리된 값의 합을 반영한다는것은 프로그람에서의 또 다른 불변량(식)이다.

```
//Program 4-6. Compute average of five numbers
#include <iostream>
#include <string>
using namespace std;
int main ( ) {
    const int ListSize =5;
    int ValuesProcessed = 0; //no Values processed yet
    float ValueSum = 0;         // no running total
    cout << "Please enter"<<ListSize<< "numbers"<<endl;
    while (ValuesProcessed < ListSize){
        // there are more Values to process
        float Value;
        cin >>Value;  // extract the next Value
        ValueSum += Value;  //add Value to running total
        ++ValuesProcessed;    //increment number of Values
        // that have been processed
    }
    float Average = ValueSum /ValuesProcessed;
    cout << "Average:"<<Average <<endl;
```

```
        return 0;
    }
```

프로그램 4-6. while문에 대한 프로그람실례

프로그람은 이 경우에 입력을 처리하여야 하므로 사용자에게 입력재촉문을 표시한다.

지금까지 처리된 값의 수가 목록의 크기보다 작으므로 고찰해야 할 값들은 많다. 이 조건은 while 순환검사식(ValuesProcessed<ListSize)에 해당된다. 만일 식이 참이면 아직 첨가해야 할 처리값들이 있으며 식이 거짓이면 구해야 할 값들이 더는 없다.

처리값은 여러개의 명령문을 요구하므로 while순환본체는 대괄호에 의해 둘러 막힌다.

```
    while (  Value Processed< ListSize ) {
        float    Value;
        cin>> Value;
        ValueSum + = Value;
        ++Value Processed ;
    }
```

본체내에서 첫번째 명령문은 입력을 얻기 위하여 순환내에서 Value라는 국부변수를 정의한다. 추출은 그때 생기며 입력값은 ValueSum에 더해 진다. 따라서 처리된 값의 수 ValueProcessed는 증가된다. 이렇게 ValueSum과 ValueProcessed에 대한 불변량들은 참으로 된다. 식(ValueProcessed< ListSize) 은 그때 재평가된다. 만일 식이 참으로 평가되면 while순환본체는 다시 실행된다. 이 처리는 ListSize 값이 처리될 때까지 계속한다. 그 시점에서 식(ValueProcessed<ListSize)는 거짓으로 평가된다.

실행은 while문다음에 있는 명령문에서 계속되는데 객체 Average를 정의하고 실행합을 처리된 값의 개수인 목록크기로 나누어 계산한다.

```
    float  Average = ValueSum/ValueProcessed ;
```

ValueSum과 ValueProcessed가 정확히 남아 있기때문에 평균이 정확히 계산되였는가를 알수 있다. 이러한 반복적인 동작으로써 유연성을 얻는다. 만일 목록크기를 계속 변경시킨다면 코드토막의 갱신은 간단하다. 즉 ListSize내용정의에서 새로운 목록크기값을 사용한다. 그것이 어떻게 동작하는가를 더 잘 리해하기 위하여 프로그람 4-6을 통하여 조사하자. 다음의 값들을 입력하여 보시오.

3, 9, 4, 7, 17

프로그람에서 처음 3개를 정의한후에 지금까지 정의한 객체들은 다음의 값을 가진다.

ListSize	5
ValuesProcessed	0
ValueSum	0.0

while검사식이 평가될 때 0이 5보다 작으므로 순환본체가 실행된다. 정수는 3이 입력명령에서 첫번째 값이기때문에 객체 Value에 3이 할당된다. 2개의 할당은 그때 제각기 ValueSum은 3으로 ,ValuesProcessed는 1로 증가시킨다. 첫번째 순환후에 객체는 다음의 값을 가진다.

ListSize	5
ValuesProcessed	1
ValueSum	3.0
Value	3.0

while검사식은 그때 재평가된다. 1이 5보다 작으므로 순환본체는 다시 실행된다. 추출은 이때 Value에 9를 할당한다(9는 입력명령에서 두번째 값이다). 할당명령문은 그때 ValueSum은 9로, ValuesProcessed는 1로 증가시킨다. 두번째 순환후 객체는 다음의 값을 가진다.

ListSize	5
ValuesProcessed	2
ValueSum	12.0
Value	9.0

while검사식은 그때 3번째로 평가된다. 2가 5보다 작으므로 순환본체는 다시 실행된다. 추출은 객체 Value에 4를 할당한다(4는 입력명령에서 3번째 값이다). 2개의 할당은 그때 제각기 ValueSum은 4만큼, ValuesProcessed는 1만큼 증가시킨다. 이 순환후 객체들은 다음의 값을 가진다.

ListSize	5
ValuesProcessed	3
ValueSum	16.0
Value	4.0

while검사식은 그때 네번째로 평가된다. 3이 5보다 작으므로 순환본체는 다시 실행된다. 추출은 객체 Value에 7을 할당한다(7은 입력명령에서 4번째 값이다). 2개의 할당은 제각기 ValueSum은 7을, ValuesProcessed는 1을 증가시킨다. 이 순환후 객체들은 다음의 값을 가진다.

ListSize	5
ValuesProcessed	4
ValueSum	23.0
Value	7.0

while검사식은 그다음 5번째로 평가된다. 4가 5보다 작으므로 순환본체는 다시 실행된다. 추출은 객체 Value에 17을 할당한다(17은 입력명령에서 5번째 값이다). 두개의 할당은 그때 제각기 ValueSum은 17을, ValuesProcessed는 1을 증가시킨다. 이 순환후 객체들은 다음의 값을 가진다.

ListSize	5
ValuesProcessed	5
ValueSum	40.0
Value	17.0

while검사식은 그다음 6번째로 평가된다. 5는 5보다 작지 않으므로 **while**명령문은 끝나고 실행은 Average의 정의를 계속하는데 8로 설정한다. 평균계산문제는 더 큰 목록 즉 처리의 유연성까지 제공하는 수자방식에 의하여 풀린다. 이 수자방식은 프로그람 4-7에 주었다.

```
//Program 4-7: Computes average of a list of Values
#include <iostream>
#include <string>
using namespace std;
int main ( ) {
    cout << doplease enter list of numbersif0<<endl;
    int  ValueProcessed = 0;
    float ValueSum = 0;
    float  Value;
    while (cin>>Value){
        ValueSum +=Value;
        ++ValuesProcessed;
    }
    if (ValuesProcessed > 0){
        float Average = ValueSum / ValuesProcessed;
        cout << "Average:"<<Average<<endl;
    }
    else
        cout << "No list to average"<endl;
    return 0;
}
```

프로그람 4-7. 임의의 목록값평균계산

이 새로운 프로그람은 특수한 방법으로서 프로그람 4-6과 다르다. 프로그람 4-6에서는 프로그람 4-7에서 객체 ValuesProcessed의 값으로부터 프로그람이 실행되는 기간에 지금까지 계산한 목록크기와 계산초기의 목록의 크기를 알아야 한다. ValuesProcessed와 ValueSum에 따르는 불변량은 변하지 않는다는것을 참고하시오. **while**검사식이 실행되는가를 결정하는 프로그람 4-7의 검사식은 (cin >> Value)이다.

추출조작값은 보통 입력원천명령에 대한 참조이다. 그러나 원천명령이 고갈되면(즉 추출할 값이 더이상 없다.) 그때 추출조작값은 0이다. 이처럼 식 (cin >> Value)는 값이 있을 때에만 0이 아니다. 따라서 추출값은 다른 값을 처리할 필요가 있는가를 결정하기 위한 기초로서 리용될수 있다. 더이상 입력값이 없다는것을 지적하기 위하여 사용자는 조작체계의 특이한 확장문자렬을 분류한다. UNIX체계에서는 문자렬이 보통 Ctrl+d이며 Dos와 Windows에 기초한 체계에서는 문자렬이 Ctrl+z이다. 현재 추출된 값 Value의 처리는 다시 값을 실행합 ValueSum에 더할것을 요구한다. 이와 류사하게 프로그람도 값이 목록의 크기와 지금까지의 ValuesProcessed를 증가시킨다. 만일 비지 않은 목록이 처리되면 평균은 계산될수 있다. 어떤 값이 추출되였겠는가를 결정하기 위하여 검사는 **if**명령문을 사용하여 진행한다.

일부 입력값이 추출되였다면 그때 ValuesProcessed는 0보다 더 크다. 만일 대신에 값없이 제공된다면 그때 추출도 없고 ValuesProcessed는 0초기값을 가진다.

break명령문은 흔히 순환을 빨리 끝내기 위하여 리용한다. 다음의 코드토막은 표준입력명령 cin으로부터 정의된 값을 찾을 때까지 입력값을 추출하고 현시한다.

```
int KeyValue;
cin >> KeyValue;
int Input;
while(cin >> Input){
    if (input != KeyValue)
        cout << Input << endl;
    else
    break;
}
```

순환에서 break명령문의 사용은 프로그람의 동작을 리해하기 더 어렵게 할수 있기때문에 흔히 쓰지 않는다. 특히 그것은 불변량을 증명하기가 더 어렵다는것을 보여 준다. 흔히 break명령문을 빼기 위하여 이러한 순환을 쉽게 다시 쓸수 있다.

```
int KeyValue;
cin >> KeyValue;
int Input;
while ((cin >> Input) && (Input != KeyValue)){
    cout << Input << endl;
}
```

4.8 간단한 문자렬과 문자처리

본문명령을 보는데는 일반적으로 2가지 경우가 있다. 한 경우 본문은 많은 문자렬로 구성되여 있다. 다른 경우 본문은 많은 문자들로 된 여러개의 행들로 구성된다. 어느 한 경우도 처리되는 자료를 모르기때문에 while구조는 대체로 본문을 처리하는데 리용된다.

주의

불완전한 while인수

while 식이 초기에 거짓으로 평가된다면 그때 while본체는 수행되지 않는다는것을 아는것이 중요하다. 초기프로그람작성자들은 흔히 본체가 한번은 동작한다고 잘못 생각한다. 프로그람은 결코 while명령문의 본체내에서 초기값을 얻는 객체에 의존하지 않는다. 실례를 들어 다음의 코드토막을 시작하기전에 표준입력명령안에 더이상 자료가 없다면 그때 추가지령은 명백히 설정되지 않은 객체를 표시한다.

```
int Number;
while (cin >> Number) {
    cout <<"Extracted another Value"<<endl;
}
cout <<"Last input:"<<Number;
```

이 절에서 처리의 두가지 실례를 준다. 목록 4-1에는 표준입력지령 cin으로부터 문자렬명령을 처리하기 위하여 3개 부분으로 구성된 표준코드토막을 준다. 첫번째 부분은 준비작업이 필요할 때마다 한다. 두번째 부분은 개별적인 문자렬들에 대하여 실지 추출하고 처리를 진행한다. 세번째 부분은 마지막후처리기능을 수행한다.

목록 4-1.　　　　　　　　　　본문처리를 위한 형태

```
//prepare for string processing
…
//extract and process strings
while (cin>> s) {
    //prepare to process string s
    …
    //process current string s
    …
    //prepare to process next string
    …
}
//finish string processing
    …
```

가운데부분은 매 문자렬을 추출하기 위하여 1번 순환하는 **while**순환으로 수행된다. 순환본체는 3개의 보조부분으로 구성된다. 첫번째 보조부분은 현재의 문자렬을 처리하기 위한 준비작업을 한다. 현재문자렬의 실제처리는 다음에 진행된다. 거기서 다음의 순환을 준비하기 위한 전처리가 수행된다.

모델토막을 리용한 증명을 프로그람 4-8과 4-9에 주었다. 프로그람 4-8은 매 추출된 문자렬이 단어라는 가정에 따라 표준입력을 시험하고 입력된 단어수와 관사수를 결정한다(즉 the, an, or, a). 실례를 들어 다음의 본문이 프로그람 4-8의 입력으로 주어 진다면

There once was a course on discourse
Where the instructor was quite course.
He mumbled. And he fumbled.
And once took a tumble.
When he realized his remarks were too course!

그때 프로그람은

Text contains 31 words of which 3 are articles

를 출력한다. 프로그람 4-8은 3개의 정의

```
int NumberOf Words = 0;
int NumberOfArticles = 0;
string s;
```

로 시작한다.

```
// Program 4-8: Computes number of words and articles
#include <iostream>
#include <string>
```

```
using namespace std;
int main() {
    // initialize counters for text analysis
    int  NumberOfWords = 0;
    int  NumberOfArticles = 0;
    // begin string processing
    string s;
    cout << "Enter text for analysis:"<<endl<<endl;
    while (cin >> s){
        //found another word
        ++NumberOfWords;
        //test if the word an article
        if ((s == "the" || (s == "The" || (s == "an"
          || (s == "An" || (s == "A" || (s == "a") {
            //the word an article
            ++NumberOfWords;
        }
    }
    //display test statistics
    cout << endl;
    cout << "Next contains "<<NumberOfWords
      << "words of which"<<NumberOfArticles<< "are articles"<<endl;
    return 0;
}
```

프로그람 4-8. 단어통계량계산

객체 NumberOfWords와 NumberOfArticles는 제각기 너무 차이나게 추출된 단어수와 그 단어들에서 관사의 수를 위하여 실행합으로 사용한다. 문자렬 s는 현재의 문자렬을 표시한다. 입력재촉문과 관계되는 이러한 정의들은 문자렬전처리부분으로 된다. 목록 4-1에서 보여 주는것처럼 while순환본체는 현재문자렬의 처리를 수행한다. 프로그람 4-8에서 이 문자렬처리는 2개의 명령문을 포함한다. 첫번째 명령문은 새로 추출된 문자렬 s가 다음단어를 의미하므로 NumberOfWords계수기를 증가시킨다.

```
++NumberOfWords;
```

문자렬 s는 그때 관사인가를 보기 위하여 검사된다. 관사는 리용된 초기문자로 나타날수도 있고 나타나지 않을수도 있으므로 s가 관사인가를 결정하는 검사식은 6개 용어 즉 매 가능한 관사에 대한 대문자와 소문자를 가진다.

만일 임의의 용어가 참이면 s가 관사이므로 용어는 or연산자로써 결합된다.

```
((s=="the")||(s=="The")||(s=="an")
      ||(s=="An")||(s=="a")||(s=="A"))
```

만일 s가 관사라는것이 발견되면 그때 계수기 NumberOfArticles는 증가된다.

++NumberOfArticles;

```
//Program 4-9 : Detects and reports duplicate strings in input
#include <iostream>
#include <string>
using namespace std;
int main( ) {
    //prepare for string processing of duplicates
    string Prev = " "     string s;
    int count=0;
    cout<<"Enter text for duplication detection"<< endl<<endl;
    //extract and process strings
    while (cin>>s){
        //process current string s
        if (s==Prev) { // s is a duplicate
            ++count;
        }
        else { //s is not a duplicate
            if (count>1){
                //Previous duplicate run needs to be
                //reported
                cout<<Prev<<"occurs"<count
                <<"Times in a row"<endl;
            }
            //s is a run of size 1 so far
            Prev=s;
            count =1;
        }
    }
    //finish up string processing of duplicates
    if (count>1) {
        //text ended with a string of duplicates
        cout<<Prev<<"0ccurs "<<count
          <<"Times in a row"<endl;
    }
    return 0;
}
```

프로그람 4-9. 중복되는 입력통보

순환이 끝나면 모든 문자렬은 추출되며 본문통계량을 표시할 준비를 한다. 표시는 적당한 표시와 함께 cout계수기 NumberOfWords와 NumberOfArticles를 추가하여 수행한다.

```
cout << "Text contains" << NumberOfWords
  << "of which" << NumberOfArticles
  << "are articles" << endl;
```

다른 문자렬증시프로그람은 프로그람 4-9인데 련속 중복되는 입력문자렬을 시험한다. 실례를 들어 표준입력이

```
We are even lonier than you you think
But we are not so funny as as we think
Ha ha ha ha
```

이라면 그때 프로그람은 다음의 해당한 정보를 표시한다.

```
"you"occurs 2 times in a row
"as"  occurs 2 times in a row
"ha"  occurs 3 times in a row
```

프로그람 4-9에 의하여 수행된 검사처리는 사실 대소문자를 구별한다. 실례를 들어 세번째 행을 시작하는 Ha는 직접 그다음에 쓰는 ha와 구별된다. 련습에서는 대소문자를 구별하지 않고 중복성을 찾는 프로그람 4-9의 어느 한 판본을 생각한다.

프로그람 4-9는 문자렬객체 Prev와 s를 정의하여 문자렬의 처리를 준비한다.

```
string Prev = " ";
string s;
```

문자렬 s는 정확히 처리되는 문자렬을 나타내며 문자렬 Prev는 s전에 직접 처리된 문자렬을 나타낸다. 프로그람을 시작하기전에 문자렬이 없기때문에 Prev는 빈 문자렬로 초기화한다. 객체 count도 또한 정의된다. 객체 count는 Prev에 의하여 표시되는 문자렬과 지금까지의 련속발생수를 나타낸다. 그것의 초기값은 0이다.

```
int  count = 0;
```

간단한 문제는 **while**순환본체에 반영된다. 현재의 문자렬 s의 전처리가 요구하는것은 현재의 문자렬 s를 위하여 주어 진 동작이 s가 전에 추출된 문자렬 Prev의 값을 중복하지 않는가 하는데만 의존한다. 만일 문자렬 s와 Prev가 같은 값을 나타낸다면 count는 증가된다.

```
if (s==Prev) {    // s is a duplicate
++count;
}
```

만일 문자렬 s와 Prev대신에 다른 값이 표시되면 몇번째 중복인가를 결정한다. count검사로써 할수 있는데 Prev에 들어 있는 문자렬값의 련속발생수를 유지한다. 만일 count가 1보다 더 크다면 중복이 있다. 추가명령문

```
cout << "\ " << Prev << "\ " occurs "cout"
```

```
      << "times in a row" << endl;
```

은 이 부분에서의 중복정보를 정확히 알린다. 만일 대신에 count가 1보다 더 크지 않다면 그때 개별적인 동작은 필요없다. count의 값에는 관계없이 현재문자렬 s는 앞으로의 중복발견을 위하여 Prev에 복사된다.

```
Prev = s;
```

추가적으로 count의 값은 1로 설정된다.

```
Count=1;
```

이렇게 count는 Prev에 의하여 유지되는 문자렬의 값의 련속발생수를 반영한다. 순환이 끝나면 중복통보는 마지막문자렬의 추출에 필요하다. 순환내에서 중복처리가 진행되도록 하기 위하여 수행된 검사는 여기서 진행한다.

```
if (count > 1) {
    // text ended with a string of duplicates
    cout << Prev<<"occurs" <<count
      << "times in a row"<< endl;
}
```

목록 4-2에 문자들의 명령처리를 위한 모델코드토막을 주었다. 목록 4-1이 아닌 방법의 한가지 우점은 이 방법이 본문의 개별적행들의 위치를 결정할수 있다는것이다. 이 발견은 그것이 입력을 문자렬로만 보기때문에 목록 4-1에서는 가능하지 못하다. 목록 4-2의 결함은 프로그람작성자가 개별적문자들의 문자렬구조체외형을 만들게 한다는것이다.

목록 4-2의 코드토막은 본문처리를 위하여 예비작업이 필요할 때마다 진행하는 부분부터 시작한다. **bool**형객체 MoreLinesToProcess는 정의되고 추가적인 처리가 필요한가를 추적하기 위하여 초기화된다.

while순환은 개별적인 행들을 처리한다. **while**순환의 반복은 MoreLinesToProcess에 의하여 조종된다. **while**본체는 현재행에서 문자들을 처리하기 위한 예비부분으로 시작한다.

bool형객체 MoreCharactersOnCurrentLine은 이 부분에서 정의되고 초기화된다. 객체의 목적은 현재행에서 추가적인 문자들을 처리하려는가를 지적하는것이다.

두 론리형객체 MoreLinesToProcess와 MoreCharactersOnCurrentLine의 값들은 본문처리에 대한 참조와 함께 불변량들이며 그것들이 참일 때 더욱 길어 지며 처리해야 할 더 많은 행들과 현재행에서 처리해야 할 더 많은 문자들이 있다. **while**순환은 개별적인 문자들을 처리하는데 사용된다. 이러한 내부 **while**순환은 처리에 필요한 매 문자를 위하여 반복한다. 순환안에 순환을 넣는것은 힘 있는 프로그람작성구조인데 그것은 내부순환이 바깥순환의 매 반복때마다 수행되므로 프로그람이 많은 고정작업을 수행하도록 할수 있다. 내부**while**의 본체는 개별적인 문자를 위한 예비부분으로 시작한다. 이 준비부분은 추출된 문자를 표시하기 위하여 **char**형객체 CurrentCharacter를 정의하고 있다.

목록 4-2. 본문처리를 위한 모형

```
//prepare for processing text
bool MoreLinesToProcess=true;
 ⋮
```

```
while(MoreLinesToProcess) {
    //process next line
    bool MoreCharactersOnCurrentLine=true;
    ⋮
    while(MoreCharactersOnCurrentLine) {
        //prepare to process next character
        ⋮
        char CurrentCharacter;
        if (cin.get(CurrentCharacter)) {
            //process CurrentCharacter on current line
            ⋮
            if (CurrentCharacter=='\n') {
                //current line has no more characters
                MoreCharactersOnCurrentLine=false;
            }
            ⋮
        }
    }
    //finish processing of current line
    ⋮
}
//finish overall processing
...
```

CurrentCharacter로써의 추출은 if명령문의 검사식에서 생긴다. 추출은 추출연산자 <<로 할뿐아니라 cin의 성원함수를 통하여 한다.

RectangleShape객체들과 같이 객체 cin은 성원함수들을 가지고 있다. cin의 성원함수 get()는 표준입력명령으로부터 다음문자를 추출하며 그것을 CurrentCharacter에 기억한다. 만일 성과적으로 되면 식(cin.get(CurrentCharacter))은 참으로 평가되며 추출이 가능하지 못하면 추출은 거짓으로 평가된다. <<보다 get( )를 사용하는 리유는 get( )가 공백을 무시하지 않는다는것 즉 함수 get( )는 문자들을 다 추출한다는것이다. 다음장에서는 cin의 다른 성원함수들을 본다.

만일 함수 get( )가 한 문자를 추출한다면 그때 그 문자가 처리된다. 처리의 개별부분은 코드토막에 포함된 특수문자와 특수과제에 의존한다. 내부순환의 처리는 CurrentCharacter가 행바꾸기를 발견했을 때 자연히 끝난다.

한번에 현재행우의 문자들은 개별적으로 처리되며 내부while순환의 추가적인 반복이 필요한가를 보기 위하여 검사한다. 만일 함수 get( )가 문자를 추출할수 없다면 그때 코드토막은 내부외부while순환이 끝났다는것을 의미하는 2개의 bool형객체를 설정한다.

외부while순환은 한번에 끝나고 마지막전처리부분이 생긴다. 이 모델토막은 프로그람 4-10에서 리용되는데 입력명령의 매행을 출력명령에 반영한다. 반영은 소문자와 같은 큰 입력문자를 표시하는 방법

으로 진행한다.

```
//프로그램 4-10. 소문자와 같은 입력처리
//프로그램 4-10. 입력된 대문자를 그와 같은 작은 문자로 변환
#include <iostream>
#include <string>
using namespace std;
int main( ){
    //prepare for processing text
    bool MoreLinesToProcess=true;
    while (MoreLinesToProcess){
        //process next line
        bool MoreCharactersOnCurrentLine=true;
        cout<<"Please type a line of text:";
        while (MoreCharactersOnCurrentLine) {
            //process next character on current line
            char CurrentCharacter;
            if (cin.get(CurrentCharacter)) {
                //process current character on current line
                if (CurrentCharacter=='＼n'){
                    //found newline character that ends line
                    MoreCharactersOnCurrentLine=false;
                }
                else if ((CurrentCharacter>='A')
                  &&(CurrentCharacter<='Z')) {
                    //CurrentCharacter is uppercase
                    CurrentCharacter=CurrentCharacter -*0*0+*0*0
                    cout<<CurrentCharacter;
                }
                else {//nonuppercase character
                    cout<<CurrentCharacter;
                }
            }
            else {//no more characters
                MoreCharactersOnCurrentLine=false;
                MoreLinesToProcess=false;
            }
        }
        //finish processing of current line
```

```
            cout<<endl;
        }
        //finish overall processing
        return 0;
    }
```

프로그램 4-10. 입력을 소문자와 같게 반영

CurrentCharacter의 처리는 CurrentCharacter가 행바꾸기문자인가, 큰 수자문자인가 혹은 행바꾸기문자도 아니고 큰 수자문자도 아닌가에 관계된다. 이러한 결정은 **if-else-if** 명령문을 사용하여 작성한다.

```
if (CurrentCharacter == 'n'){
    // found newline character that ends line
    MoreCharactersOnCurrentLine = false;
}
else if ((CurrentCharacter >= 'A') && (CurrentCharacter <= 'Z')){
        //CurrentCharacter is uppercase
        CurrentCharacter=CurrentCharacter-'A'+'a';
        cout << CurrentCharacter;
}
else { //nonuppercase character
        cout << CurrentCharacter;
}
```

만일 CurrentCharacter가 ASCII 'A'부터 'Z'사이에 놓이면 그것은 큰 문자이다. 큰 문자는 대문자의 값과 'a'의 서로 다른 값을 더한데로부터 'A'값을 덜어 소문자로 변환할수 있다 (CurrentCharacter가 얼마나 차이나는가 하는 편위주소의 서로 다른 값은 수자의 시작부터이다).

```
CurrentCharacter=CurrentCharacter-'A'+'a';
```

문 제

34. 다음의 코드토막을 분석하시오.

```
int I= 1;
while ( I < n) {
    if ((I % 2) ==0){
        ++I;
    }
}
cout<< I << endl;
```

n이 9이라면 무엇이 출력되는가?

35. 다음의 코드토막을 분석하시오.

```
int I = 1;
```

```
while (I<n){
    if ((I%2)==0) {
        ++I;
    }
}
cout << I << endl;
```

n이 10이라면 무엇이 출력되는가?

36. 다음의 코드토막을 분석하시오.

```
int I=1;
int j=3;
while(I<15 && j<20){
    ++I;
    j+=2;
}
cout << I+j << endl;
```

무엇이 출력되는가?

37. x의 값이 2부터 40사이가 되게 하기 위하여 방정식 x^2+x+49의 값을 발생시키는 while순환을 작성하시오.
38. cin명령으로부터 본문을 읽는 프로그람을 작성하고 입력에서 수자(즉 '0', '1', '2', '3', '4', '5', '6', '7', '8', '9')의 개수를 계수하시오.

4.9 for명령문에 의한 순환

while구조가 모든 반복요구를 조종하는 편리한 유연성을 가지지만 C++는 2개의 또 다른 반복구조 for와 do를 제공한다. 이 구조들은 일부 일반프로그람작성을 더 쉽게 해준다. 흔히 더 많이 사용되기때문에 먼저 for구조를 생각해 보며 그다음 간단히 do구조를 생각한다. for구조는

와 같은 형태를 가지는데 forInit는 객체정의가 아니면 식이며 forExpression 그리고 PostExpression은 생략될수 있다. 그림 4-8에 for구조를 위한 흐름도를 주었다. for명령문이 프로그람안에 있을 때 forInit가 먼저 수행되고 그다음 forExpression이 평가된다(forExpression이 제공되지 않으면 대신에 값이 포함되여 리용된다는것을 참고하시오). 만일 forExpression이 참이면 동작은 실행되며 그때 PostExpression이 실행된다. forExpression은 그때 재평가되며 다시 참이면 동작의 실행과

PostExpression이 반복된다.

경험

효과적인 조건식쓰기

객체가 실지수값안에 있는가를 검사하는 조건식은 흔히 복잡하다. 이것은 수값범위안에 있는가 하는 검사를 수자순서대로 하여 간단하게 만들수 있다. 실례를 들어 while명령문

```
while (NumberOfElements> i) {
    // Process the ith element
    ...
}
```

를 생각해 보시오. 의미는 NumberOfElements보다 i가 작다는것이다.

i가 numberOfElements보다 항상 더 작은데로부터 검사에서 먼저 나타난다.

```
while ( i < NumberOfElements ) {
    //Process the ith element
    ...
}
```

이 안내행은 실지로 두 값이 범위경계에 놓이게 될 때 효과적이다. MinElement<i <MaxElement인가를 보기 위한 경우를 상상해 보시오. 다음의 어느 경우가 더 명백한가?

```
while (i<Maxelement && i >MinElement {
    ...
}
while (MinElement< i && I < MaxElement){
    ...
}
```

실지로 두번째것이 더 명백하다. 그것은 식에서 요소의 순서가 검사되는 값의 순서와 갈기 때문이다.

forExpression의 검사와 동작의 실행 그리고 PostExpression은 forExpression이 거짓으로 평가될 때까지 계속된다. 그때 평가는 프로그람에서 다음의 명령문으로 계속한다. 만일 forExpression이 초기에 거짓이라면 그때 동작도 PostExpression도 수행되지 않으며 프로그람은 직접 다음명령문을 계속한다. 대부분의 프로그람작성자들은 for명령문에 대하여 필요한 초기화를 진행하는 forInit와 for명령문본체의 다음순환에 대하여 준비하는데 필요한 PostExpression을 사용한다.

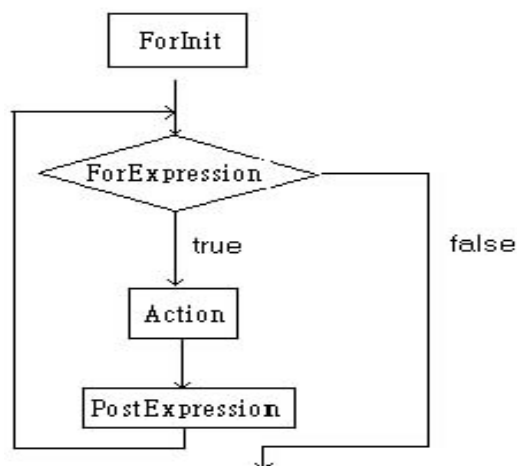



그림 4-8. for구조의 흐름도표시

다음실례에서는 옹근수 1부터 n까지의 적을 계산한다. 수학적으로 이 값을 n!이라고 하며 그 정의는

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n*(n-1)*\cdots*1 & \text{if } n>=1 \end{cases}$$

이다.

다음의 코드토막은 먼저 대기하여 옹근수값 n을 얻는다. 그때 요구하는 차례곱을 계산하기 위하여 **for**명령문을 사용한다. 여기서는 간단히 성과적으로 하기 위하여 입력형이 정확한가를 확인하지 않는다.

```
cout << "Please enter a positive integer:";
int n;
cin >> n;
int nfactorial = 1;
for (int I=2;I<=n;++I){
    Nfactorial*=I;
}
cout << n << "!=" << nfactorial << endl;
```

입력명령이 실행될 때 변수 n에 값 4를 넣는다고 하자. 코드토막은 그다음 nfactorial를 정의하고 1로 그것을 초기화한다. 이 객체는 다음의 값을 가진다.

N	4
Nfactorial	1

for명령문은 정의하는 forInit로 시작하며 객체를 값 2로 초기화한다(관계되는 첫번째 요소). 전통적인 프로그람술어에서 i는 색인변수라고 하는데 그 값은 매 순환에 따라 변화된다. i의 값은 차례곱계산에서 사용된 현재요소를 반영한다. 요소들이 증가순서로 발생되므로 nfactorial은 항상 1부터 n-1의 n개 요소의 적을 반영한다. 이 동작은 불변량이다.

nfactorial을 현재요소 i로 적당히 변경하고 그다음 i를 다음번 가장 큰 요소로 변경시키면 계산은 성과적이다. 이 점에서 객체들은 다음의 값을 가진다.

n	4
nfactorial	1
I	2

for명령문은 그다음 j를 n과 비교한다. 2가 4보다 작으므로 **for**순환본체는 실행된다. 본체는 nfactorial을 i에 의하여 계산한다. 결과는 객체 nfactorial이 값 2를 가지는것이다.

식 ++i는 그것이 3이 되게 하기 위하여 1을 증가시키는 색인 i를 발생시켜 그때 평가한다.

n	4
nfactorial	2
I	3

그다음 순환은 식 i<n을 재평가한다. 3이 4보다 작으므로 **for**순환본체는 다시 실행된다. 본체에서 객체 nfactorial은 다시 색인 i에 의하여 계산된다.

결과는 객체 nfactorial이 값 b를 가진다는것이다. 식 ++i는 그것이 값 4를 가지게 하기 위하여 1을 증가시키는 색인 i를 발생시켜 그때 평가한다.

n	4
nfactorial	6
I	4

그다음 순환은 식 i<=n을 재평가한다. 4는 4와 같으므로 for순환본체는 다시 실행된다. 색인 i에 의한 nfactorial의 계산은 값 24를 준다. 식 ++i는 값 5를 가지기 위하여 1을 증가시키는 색인 i를 발생시켜 평가한다.

n	4
nfactorial	24
I	5

순환은 그다음 식 i<=n을 재평가한다. 5는 4보다 작거나 같지 않으므로 for명령문은 끝난다. 실행은 명령문

```
cout<<n<<"!="<<nfactorial<<endl;
```

로 계속되는데 다음의 출력을 만든다.

```
4!=24
```

만일 객체 n이 0 혹은 1이라면 forExpression i<=n이 처음 거짓으로 되는것을 조사하시오. 결과에서 nfactorial의 값은 1인데 이것은 0!과 1!에 대한 정확한 값이다.

경험

while명령문을 사용하여 for명령문을 표시

임의의 for명령문

```
for (forInit;forExpression;PostExpression)
Action;
```

은 while명령문으로 바꿀수 있다. 실례를 들어 다음의 코드토막은 우의 for명령문과 같다.

```
{ forInit;
    while (forExpression) {
        Action;
        PostExpression;
    }
}
```

바깥괄호는 정의 forInit의 범위를 제한하는데 필요하다.

프로그람 4-11에서는 목록의 입력값평균을 계산하는 문제를 더 빨리 풀기 위한 반복적인 구조로서 for명령문을 리용한다. 이 프로그람에서 리용된 객체들은 프로그람 4-7에서 볼수 있는것처럼 같은 역할을 수행하며 같은 불변량들은 그 값들에 따라 리용된다. ValuesProcessed의 초기화는 for구조의 초기화식이다. C++표준은 for순환의 forInit에서 정의된 객체들이 for명령문에서만 사용될수 있다는것을 보여 주므로 ValuesProcessed는 for명령문을 선행한다.

```
//program 4-11 :Compute the average of an arbitrary
//list of numbers
#include <iostream>
#include <string>
using namespace std;
int main( ) {
    cout<<"Please enter list of numbers"<endl;
    float ValueSum=0;
    float Value;
    int ValuesProcessed;
    for (ValuesProcessed=0;cin>>Value;
      ++ValuesProcessed) {
        ValueSum+=Value;
    }
    if (ValuesProcessed > 0) {
        float Average = ValueSum/ValuesProcessed;
        cout <<"No list to average"<<endl ;
    }
    else
    cout<< "No list to average" <<endl ;
    return 0 ;
}
```

프로그램 4-11. for명령문을 리용한 수렬의 평균계산

만일 for명령문안에서 ValuesProcessed를 정의하면 평균계산에서 순환후 그것을 사용할수 없다. for명령문을 리용하는 다음실례는 피보나치수렬이라는 중요한 수학부분을 포함한다. 6장에서 더 많은 세부로서 이 수렬을 본다. 수렬은 다음의 수 1, 1, 2, 3, 5, 8, 13, 21로 시작한다. 초기의 두 수 1다음에 오는 수렬의 매수는 2개의 이전 수들의 합이다. 실례를 들어 1+1=2, 1+2=3, 2+3=5, 3+5=8 등이다. 다음의 코드토막은 값 n을 얻는다. for순환은 그때 수렬에서 먼저 n의 수를 표시하는데 리용된다. 그것은 n을 2로 가정한다.

```
cout <<"Please enter an integer greater than 2:";
int n;
cin>> n;
cout<<"The first"<<n<<"피보나치 numbers:"
    <<endl<<1<<endl<<1<<endl;
int Previous Number=1;
int CurrentNumber=1;
for (int I=3;I<=n;++I){
    int sum=PreviousNumber+CurrentNumber;
```

```
        cout<<sum<<endl;
        PreviousNumber=CurrentNumber;
        CurrentNumber=sum;
    }
```

N의 추출다음에 코드토막은 수렬에서 처음 두 수를 표시한다. 다음객체 PreviousNumber와 CurrentNumber를 정의한다. 그것들은 수렬에서 이전에 처리된 2개의 수들을 표시한다. **for**순환은 그때 n-2번 순환한다. 객체 i는 순환색인변수이며 그것은 3부터 n사이의 값을 가진다. 매 순환방법은 값이 이전에 처리된 두 수의 합인 객체 sum의 정의와 표시에 의하여 시작된다. 그다음 다른 수가 처리된다는 것을 반영하여 PreviousNumber와 CurrentNumber의 값을 변경시킨다. 다음코드토막에서 사선형식으로 3개의 정방형 RectangleShape객체들을 표시하는 **for**순환을 사용한다.

코드토막의 출력을 그림 4-9에 주었다.

```
SimpleWindow W("One diagonal",5.5,2.25);
W.open( )
int I=0;
for (int j=1;j<=3;++j){
    RectangleShape S(W,I+j*0.75+0.25,
    j*0.75-0.25,Blue,0.4,0.4);
    S.Draw( );
}
```

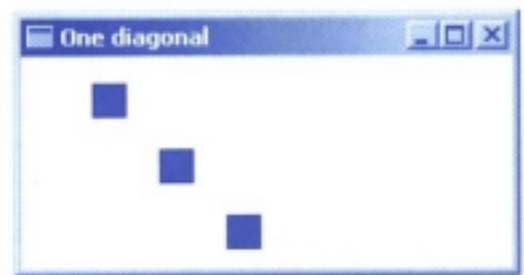



그림 4-9. 사선으로 정방형표시

사선생성토막은 한편의 과제를 위한 적당한 표시와 크기를 가진 SimpleWindow객체 W정의에 의하여 시작한다. 객체 i는 그때 0으로 정의되고 초기화된다(다음실레는 사선으로 3개의 4각형을 그리기 위하여 이 코드토막의 발생에서 i를 사용한다). **for**순환은 색인변수로서 j를 사용한다. 실지로 색인 j는 값 1, 2 그리고 3으로 주어 진다. 색인변수는 그것이 1로 초기화되고 매 순환후 1을 증가시키므로 이 값들로 주어 진다. Window W, 객체 I 그리고 색인 j는 RectangleShape S의 정의에서 사용된다. 객체 s는 순환을 통하여 매번 재정의된다. 매 정의는 Window W에 면이 0.4cm인 푸른 정방형 RectangleShape를 할당한다. 첫 순환에서 정의된 RectangleShape의 중심은 자리표 (I+j*0.75+0.25, J*0.75-0.25)=(1, 0.5)에 있다. 두번째 순환에서 정의된 RectangleShape의 중심은 (1.75, 1.25)이다. 세번째와 마지막순환에서 정의된 RectangleShape의 중심은 (2.5, 2)이다. 이렇게 순환을 통해 매번 s는 창문 W에서 서로 다른 위치에 생기게 하기 위하여 재정의되고 다시 그려 진다. 그러나 s의 이전 판본은 표시부분을 얻는다. 이 동작은 창문체계의 문자인데 객체는 여러가지 지우기지령을 활발히 발동하여서만 표시에서 지울수 있다. 다음의 코드토막은 다른 **for**순환안에서 이전 코드토막의 순환을 넣는다.

```
SimpleWindow W("Three diagonals",5.5,2.25);
W.open( );
for (int I=0;I<=2;++I){
    for(int j=1;j<=3;j++){
        RectangleShape S(W, I+j*0.75+0.25, J*0.75-0.25, blue, 0.4,0.4);
        S.Draw( );
    }
```

}

외부for순환은 색인 i를 사용하는데 실지값 0, 1 그리고 2를 가진다. 이 순환이 3번 순환하므로 내부for순환은 3번 실행되며 내부순환의 매 실행에서는 3개의 정방형을 그리므로 총 9개의 정방형을 그린다. 이 토막의 실행출력을 그림 4-10 에 주었다.

바깥for순환이 초기화되면 i는 값 0이 되므로 첫 3개 4각형은 중심 (1, 0.5), (1.75, 1.25) 그리고 (2.5, 2)를 가진다(이전 실례에서와 같이). 바깥for순환의 두번째 순환에서는 i가 값 1을 가진다. 이것은 내부 for순환이 중심 (2, 0.5), (2.75, 1.25) 그리고 (3.5, 2)에 4각형을 그리게 한다. 바깥for순환의 마지막순환은 3개의 4각형을 더 그린다. 그것들은 중심이 (3, 0.5), (3.75, 1.25) 그리고 (4.5, 2)이다.

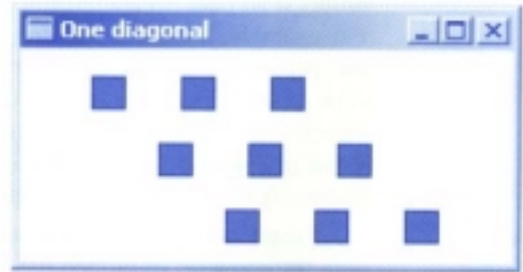



그림 4-10. 세 사선에 따라 9개의 4각형그리기

주의

무한순환

모든 while과 for실레는 끝을 가진 순환명령문을 만드는 여러가지 동작을 가진다. 끝 없는 이 프로그람은 무한순환으로 되는데 그것들은 조작체계지령이 저절로 끝날 때까지 순환을 계속한다. 프로그람에서 이 문제를 피하기 위하여 순환명령문에서 하려는것을 확고히 리해하시오. 개별적으로 순환명령문안에 끝내는것이 설계되여 있다는것을 확인하시오.

4.10 간단한 자료표시

여러 현상에 대한 15번의 조사를 표시하는 다음과 같은 자료모임이 주어 졌다고 가정하자. 무슨 명령문을 만들수 있는가?

4.90 2.41 0.82 0.77 2.60 5.10 7.52 9.45
9.65 7.81 5.04 2.51 0.95 0.80 2.62

통계해석수행으로 만들수 있는 한개 명령문은 검사된 증상이 평균값 4.97과 함께 0…10사이로 나타난다는것이다. 앞으로 통계해석을 수행할수 있으며 통보할수 있다. 실례로 조사의 표준편차는 2.95이다. 증상을 리해하기 위한 다른 한가지 방법은 조사렬이 패턴을 표시하는가를 알아 내는것이다. 실례를 들어 수자들이 작아 지게 혹은 피보나치형순서로 배렬되는가?

만일 패턴을 알지 못한다면 자료표시를 해야 한다. 실레를 들어 2차원적방법으로 자료모임값을 계산하는 일반적인 방법은 자료모임수렬에서 자료모임값에 해당한 y축과 위치에 해당한 x축을 가지는것이다(다시 말하면 처음 두 자료모임값은 점 (1, 4.9)와 (2, 2.41)에 해당한다). 이러한 자료모임에 대한 계산을 그림 4-11에 주었다. 계산은 구조를 가지는 수들 즉 시누스선형식으로 나타나는 수들을 의미한다. 프로그람 4-12는 수자료모임을 해석하기 위한 계산점모임으로 변환하는 간단한 자료표시도구이다. 그것은 그림 4-11과 같은 계산결과를 주었다. 프로그람 4-12는 EzWindows, RectangleShape클라스를 리용하여 반복적으로 4각형을 그리면서 과제를 수행하는데 i번째 4각형에서는 자료모임에서 x자리표가 i이고 y자리표가 i번째 값인 위치가 중심으로 된다.

프로그람은 상수형Unit를 정의하는것으로부터 시작한다. 매 계산점은 길이 Unit의 면들을 가진 4각형을 그린다.

```
const float Unit=0.25;
```

자료모임의 크기 n은 그때 추출된다.

```
cout<<"Size of data set:";
int n;
cin>> n;
```

그다음 프로그람은 정의하고 자료모임계산을 포함하는 SimpleWindow객체 W를 연다.

```
SimpleWindow W("Data set display",n+2,10);
W.open( )
```

창문의 크기는 n+2*10cm로 고정된다(련습에서 보다 완성된 프로그람을 개발한다).

그림 4-11. 점형태로 결합된 수들로써 자료모임의 계산

```
//program 4-12:Simple visualization tool
#include <iostream>
#include <string>
#include "rect.h"
using namespace std;
int ApiMain( ) {
    const float Unit=0.25;
    cout<<"Enter size of data set:";
    int n;
    cin>>n;
    SimpleWindow W("Data set display",n+2,10);
    W.open( );
    for (float x=1;x<=n;++x) {
        cout<<"Enterdata Value(n) :";
        float y;
        cin>>y;
```

```
            RectangleShape Point(W,x,y,Blue,Unit,Unit);
            Point.Draw( );
        }
        return 0;
}
```

프로그람 4-12. 자료모임값계산

for순환은 자료모임을 보여 준다. 순환은 n번 진행된다. 색인 x는 순환회수를 계수하기 위하여 리용한다. 매 순환에서 순환본체는 먼저 다음 자료모임값 y를 얻는다.

```
cout<<"Enter data Value(n):";
float y;
cin >> y;
```

RectangleShape객체 Point는 그다음 정의되고 창문 W에 그려 진다. Point는 길이 Unit인 면으로 되고 중심위치가 (x,y)인 푸른 4각형을 표시한다.

```
RectangleShape Point(W,x,y,Blue,Unit,Unit);
Point.Draw( );
```

Point는 한번 그리고 for순환의 표현식을 평가하며 i를 증가시킨다. 증가는 다른 자료모임값이 처리된다는것을 의미한다. 그다음 얻어 처리하여야 할 다른 자료모임값이 있는가를 결정하기 위하여 순환검사식을 재평가할 준비를 한다.

4.11 수수께끼풀기

다음은 지난 19세기로 돌아 간다(이 이야기는 뉴욕타임스의 수수께끼편집자인 윌 쇼트가 쓴것이다).

한때 나라를 떠돌아 다니는 4명의 방랑자가 있었다. 그들은 려행기간 돈을 적게 썼으며 그리하여 여러가지 직업을 찾아 한 농장에 들렸다. 농장주는 그때 여러주이상 일할수 있는 200시간짜리 일감이 있다고 했다. 농장주는 그들에게 어떻게 일을 분담하겠는가는 자신들이 결정하라고 말하였다. 방랑자들은 다음날 일을 시작하기로 하였다. 다음날 아침 네명중 제일 게으른 한 방랑자는 자기들이 같은 량의 일을 하여야 할 리유가 없다고 말하였다. 그리하여 방랑자들은 그의 제안에 따라 모두 주패를 뽑았다. 주패에 표시된 수자는 자기들이 일해야 할 날자수와 매일 일할 시간수를 의미하였다. 실례를 들어 주패에 3이라는 수가 표시되여 있으면 그것을 뽑은 방랑자는 하루에 3시간씩 3일동안 일해야 한다. 게으른 방랑자가 이 계획을 다른 세 방랑자가 리해하도록 납득시키고 숙련된 솜씨로 가장 좋은 주패를 뽑은것은 두말할것도 없다. 수수께끼는 이 계획에 따라 작업을 분담하는 가능한 방법을 결정하는것이다.

수수께끼의 결과는 $a^2+b^2+c^2+d^2=200$으로 되는 4개의 수 a, b, c, d로 구성된다. 따라서 이 4개의 수로 조합을 만들고 현재 조합된 수들의 2제곱의 합이 200이 되는가를 검사하는것이다. 수가 15이면 그의 두제곱이 200을 초과하므로 a, b, c, d는 1과 14사이에 있어야 한다. 이것보다 더 좋은 조합을 찾을수 없으므로 중복을 없애는 방향에서 조합을 만든다. 중복이 없게 조합을 만드는 쉬운 방법은 증가순서로 조합을 만드는것이다. 결과에서와 같이 매 발생된 조합은 a<=b<=c<=d와 같은 속성을 가진다. 수수께끼를 풀기 위하여 1부터 14사이에 a가 놓일수 있는 가능성을 따져 본다. a의 주어 진 값에 대하여 우에서 제기된 조합으로부터 b, c, d의 모든 가능성을 고려한다. 구체적으로 b를 위한 가능성은 a부터 14이다.

이와 같이 a와 b가 결정된 조건에서 c와 d를 위한 모든 가능성을 고려한다. c가 가질수 있는 값은 b부터 14사이이다. 한편 a, b, c가 결정되였으므로 d가 가질수 있는 값은 c부터 14이다. 이 방법은 프로그람 4-13에서 수행된다. 프로그람은 a, b, c, d를 색인변수로 하는 4개의 겹친 for명령문으로 구성된다.

프로그람의 출력은 다음과 같다.

```
Lazy hobo possible solutions
      2   4    6    12
      6    6    8    8
```

```
//Program 4.13:Display solution for lazy hobo //riddle
#include <iostream>
#include <string>
using namespace std;
int main( ) {
    cout<<"Lazy hobo possible solutions"<<endl;
    for (int a=1;a<=14;a++) {
        for (int b=a;b<=14;b++) {
            for (int c=b;c<=14;c++)
                for(int d=c;d<=14;d++) {
                    if(a*a+b*b+c*c+d*d==200) {
                        cout<<a<<""<<b<<""<<c
                           <<""<<d<<endl;
                    }
                }
            }
        }
    }
    return 0;
}
```

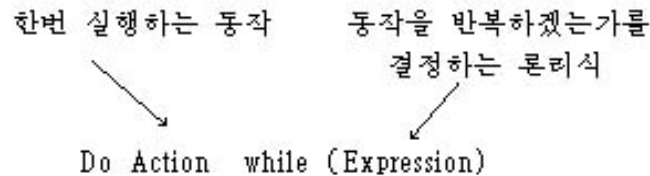
프로그람 4-13. 《게으른 방랑자》수수께끼풀기

프로그람실행과정에 고려한 가능한 결과의 개수를 결정하기는 재미 있다. 이 과제는 if명령문이 평가되는 회수를 추적하도록 프로그람을 수정하여 할수 있다. if명령문이 2,380번 실행되였다는 결론이 떨어 진다. 이 수는 크지만 14*14*14*14=38,416에 비해 작으며 증가순서로 단일조합을 만들지 않으면 안되는 렬의 수이다.

a×a+b×b+c×c+d×d가 200보다 더 클 때 내부for순환을 끝나게 하여 프로그람의 효과성을 증명할수 있다.

4.12 do명령문에 의한 순환

때때로 동작을 여러번 수행해야 하는 경우가 제기된다. 이것을 효과적으로 보기 위하여 프로그람작성자는 do명령문을 사용할수 있다.

do명령문에서 Expression은 론리식이며 Action은 다시 하나의 명령문 혹은 대괄호로 둘러 막힌 명령문들의 모임이며 다음과 같은 형태를 가진다.

구조는 Action실행으로 시작한다. Expression은 그다음 평가된다. 만일 Expression이 참이면 그때 Action이 반복된다. 이 처리는 Expression이 거짓이 될 때까지 계속된다. 그림 4-12에 흐름도를 주었다.

do는 때때로 사용자가 입력재촉문에 응답하는 처리에서 리용된다. 실례를 들어 다음의 코드토막은 반복적으로 대기하고 그다음 대답이 yes 혹은 no를 나타낼 때까지 한 문자대답을 얻는다. 만일 대답이 없으면 코드토막은 no로 가정한다.

```
char reply;
do {
     cout<<"Decision(y,n):";
     if (cin>>reply)
          reply=tolower(reply);
     else
          reply='n';
 }while ((reply!='y') && (reply!='n'));
```

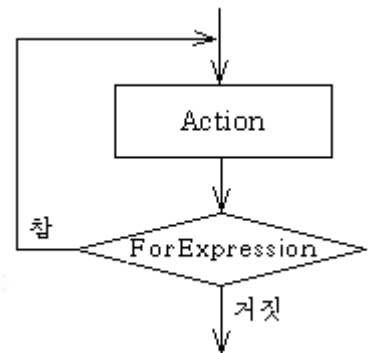



그림 4-12. do-while명령문의 흐름도

이 과제의 처리를 간단히 하기 위하여 대답을 프로그람 4-10의 변환을 모방하기보다 ctype서고함수 tolower( )를 리용하여 소문자로 변환한다. 서고함수에 대해서는 다음장에서 취급한다.

문 제

39. ArraySize>j>0이 참으로 되는 조건식을 리해하기 쉽게 작성하시오.
40. 다음의 코드토막을 분석하시오.

```
int n = 0;
for (I=0;I<10;++I)
      for (j=0;j<I;++j)
                ++n;
cout << n<<endl;
```

무엇이 출력되는가?

41. 다음의 코드토막을 분석하시오.

```
int n=10;
for(I=0;I<n;I+=2){
++n;
}
cout << I<< endl;
```

무엇이 출력되는가?

42. 다음의 코드토막을 분석하시오.

```
int n=10;
for(I=1; ; ++I){
if ((I%5)==0)
break;
else
++I;
}
cout<< I<< endl;
```

무엇이 출력되는가?

43. 다음의 코드토막을 분석하시오.

```
int n=10;
for(I=0; I<n; ){
    ++n;
    I+=3;
}
cout<<I<<endl;
```

무엇이 출력되는가?

44. 다음의 코드토막을 분석하시오.

```
int n=10;
for (I=0;I<n;++I){
    if((I%5)==0)
    break;
}
cout<<i<<endl;
```

무엇이 출력되는가?

45. 다음의 코드를 분석하시오.

```
int n = 10 ;
```

```
for (i = 0 ; i<= n ;i +=2 ) {
    if(( i % 3 ) == 0)
    break ;
}
cout << i << endl ;
```

46. 다음의 코드를 분석하시오.

```
int n = 0 ;
for(i = 0 ; i < 10 ; ++i )
    for (j = 0 ; j < i ; ++j ) {
        if (i = = 5 )
            break ;
        ++n
    }
```

계산의 력사

차분기관

콤퓨터시대에로의 전환에서 기본리정표는 19세기초에 생겨 났다. 그때 큰 문제의 하나는 과학자들과 수학자들에게 필요한 정확한 계산기와 여러가지 표의 복사였다. 측량사, 은행업자, 항해가 그리고 기사들은 이 표에 의존하였기때문에 정확성은 매우 중요한 문제였다. 문제는 표의 계산과 표시를 손으로 하였으므로 매우 지루한것이였다.

챨스 바베지(1792-1871)라는 젊은 수학자는 이 문제해결을 위한 한가지 생각을 하였다. 그의 생각은 표계산에 기계를 사용하는것이였다. 차분기관(Difference Engine)이라고 하는 이 방법은 명령에 의하여 동작하며 모든 계산을 장치적으로 수행하고 그 결과를 금속판우에 기록하는것이였다. 금속판은 표를 표시하는데 리용되였으며 결과 복사의 가능성제거와 오유표시에 리용될수 있었다.

차분기관의 수학표계산을 자동적으로 하는 방법을 리해하자면 1800년대의 표계산방법을 간단히 보아야 한다. 대부분의 수학, 물리함수들은 $dx^n+\cdots+cx^2+bx+a$형식의 다항식을 평가하여 근사화할수 있었다. 여러가지 값에 대하여 이러한 다항식들은 서로 다른 표들을 계산하여 평가할수 있다. 실례를 들어 2차다항식 $n^2+2n+22$와 다음의 표를 보고 생각해 보자. 표에서 마지막행은 식을 평가하지 않고 쉽게 계산할수 있다.

N	$n^2+2n+22$	$Difference_1$	$Difference_2$
0	22	—	—
1	25	3	—
2	30	5	2
3	37	7	2
4	46	9	2
5	57	11	2
6	70	13	2
7	85	15	2

Difference$_2$의 값은 항상 2이다. Difference$_1$의 n행값은 이전 행에서 Difference$_1$과 Difference$_2$의 합이다. 이처럼 7행통로는 85, 15 그리고 2이다. 일반적으로 n차다항식은 n개 차이를 계산하여 평가할수 있다. 차분기관은 6차다항식을 조종하여 설계된다. 바베지는 차분기관의 구조를 실현할수 있는 자금이 요구된다는것을 알았다. 그는 또한 엔진의 구조를 리용하는데 높은 기술의 제조방법이 요구된다는것을 알았다.

그는 장치의 제작에 필요한 자금을 위하여 이 문제를 해결할수 있는 대상을 찾았다. 런던의 전문회사인 Royal Society와 British requirements로부터 바베지는 1500딸라(1823년에 거의 7500딸라)를 확보할수 있었다. 이것으로서 첫번째 문제가 해결되였다. 바베지는 다음으로 제조방법을 완성하기 위하여 두번째 문제해결에 착수하였다. 그는 기계부분조립에 더 좋은 방법이 필요하다는것을 발견하였으며 여기에 자기의 노력을 다 하였다. 비록 이러한 노력이 기계부분평가에서 기능상태를 전진시켰지만 기관의 구조를 전진시키지는 못했다.

바베지가 한해동안에 두 아이와 처를 잃은것으로 하여 계획은 난관을 겪었다. 그러나 6자리수정확도로 2차원다항식을 풀수 있었다. 공교롭게 바베지는 완전한 차분기관을 완성하지 못했다. 그는 큰 기계 즉 해석기관에 대하여 상상하였으며 자기 일생을 거기에 바치기로 하였다. 해석기관은 일반적인 목적으로 설계되는데 그것은 수학함수를 계산할수 있는것이였다. 바베지의 해석기관의 개발에 많은 협조가 필요하였지만 그의 요구는 거절되였다. 바베지는 마지막까지 해석기관의 설계에 대한 작업을 계속하였다. 차분기관이나 해석기관의 제작에서 실패했음에도 불구하고 바베지의 능력은 과소평가되지 않았다. 이 두 기관에 포함된 사상은 큰 의의를 가지고 있었다. 실례를 들어 해석기관은 본질적으로 오늘의 현대적콤퓨터부분들의 모든 구성요소를 가지고 있다. 즉 착공카드로 된 프로그람이다. 그것은 기억기와 오늘의 콤퓨터처리장치에 대한 개념과 같은 계산을 수행하는 부분을 가지고 있었다.

4.13 알아 둘 점

- ✓ bool형은 c++에서 론리값을 표시할 때 쓴다.
- ✓ 론리식은 식값이 0아닌 옹근수값이거나 bool형참이면 참으로 평가된다.
- ✓ 론리식은 식값이 옹근수 0이거나 bool형거짓이면 거짓으로 한다. 론리식은 론리연산자 &&, ||, 그리고 !로 표현된다.이 연산자들은 제각기 and, or 그리고 not에 해당한다.
- ✓ 진리값표는 론리연산의 통계값이다. 표는 매개의 가능한 론리연산수들의 결합을 의미한다.
- ✓ 관계연산자도 론리값을 만든다. 관계연산자는 2개 부류 즉 동일성과 배렬로 구분한다.
- ✓ 동일성연산자 ==와 !=, 그리고 배렬연산자 <, <=, > 그리고 >=는 모든 원리와 지적자형 그리고 string클라스를 위하여 정의된다.
- ✓ 정확도에 제한이 있기때문에 류점수값은 대체로 정확도를 검사하지 않는다. 대신에 검사는 류점수값이 거의 같은가를 결정하게 한다.
- ✓ 평가되는 론리식은 간단한 연산원리에 대한 문제이다. 이 원리는 모든 표현식의 값이 평가중지를 알려 준다는것이다. 즉 p||Q에서 p가 참이면 모든 식이 참이기때문에 Q가 평가되지 않으며 P&&Q에서 P가 거짓이면 그때 Q는 모든 식이 거짓이므로 평가되지 않는다.
- ✓ if명령문은 2개의 형태를 가지고 있다. 두가지 형태에서 론리식은 평가되며 식이 참이면 동작이 실행된다. 또한 평가식이 거짓일 때 그 동작은 실행되지 않는다.
- ✓ 조건이나 순환을 가지는 구조체에 대한 동작과정을 결정하는 식은 검사식으로 표시된다.
- ✓ switch명령문은 통합식의 값에 기초하여 동작을 준다. 프로그람작성자는 그 식에 대하여 흥미 있는

부분값을 정의하며 매 부분값에 따라 요구하는 동작이 정의된다. 스위치명령문은 정의되지 않은 값을 조종하는 임의의 default부분을 허락한다.

- ✓ 프로그람작성자에 의하여 정의된 형은 형을 정의하거나 끌어 내는것과 같다.
- ✓ enum명령문은 하나의 형으로 통합내용의 모임을 구성하는 방법이다.
- ✓ 순환은 순환구조체를 사용하여 동작을 반복하는 명령들의 모임이다.
- ✓ while명령문은 주어 진 론리식이 참으로 평가되는 동안 동작을 반복한다. 만일 론리식이 초기에 거짓이면 그때 구조체동작은 결코 실행되지 않으며 동작은 검사식이 거짓으로 평가될 때까지 련속 실행된다.
- ✓ while검사식이 초기에 거짓이면 while순환본체를 실행할수 없다.
- ✓ do명령문은 while명령문과 일반적으로 류사하지만 그 동작은 적어도 한번 실행된다. 본체가 실행된 후까지 검사식은 평가되지 않기때문에 이 속성을 가진다.
- ✓ for명령문은 검사식과 첫 순환초기화동작과 순환본체의 매 실행기간 한번 실행되는 동작을 가지는 while구조체의 정의이다. for명령문의 모든 부분은 임의적이다. 개별적으로 검사식이 생략되면 그 대신에 참값이 사용된다.
- ✓ for순환의 forInit부분에 정의된 객체는 그 순환에서만 사용될수 있다.
- ✓ break명령문에 의하여 탈퇴명령문을 포함하는 switch, while, for 혹은 do조종구조체가 생긴다.
- ✓ 프로그람의 다중부분에서와 같이 순환은 세심히 진행된다. 순환전과 그사이에 주어 진 동작은 순환검사식을 항상 느끼게 한다. 이 동작은 항상 순환이 저절로 끝나게 한다.
- ✓ 불변량은 항상 참인 규칙이다. 순환개발에서 설계자는 순환에 영향을 줄수 있는 불변량에 대하여 안다.
- ✓ typedef명령문은 존재형을 위한 새 이름을 만든다. 새 이름과 낡은 이름은 련속정의에서 사용될수 있다.

련습문제

4.1 다음의 정의가 옳은가를 생각해 보시오.

```
bool P = true;
bool Q = false;
bool R = false;
string s = "a";
string t = "b";
int i = 10;
int j = 0;
```

이때 다음의 값을 평가하시오.

```
!p = Q
s != t
R == (P && Q)
Q == (P || Q)
Q || (P && !R)
P && !Q && !R || (P ==!Q)
S < t
```

```
S + t << t
i * j
i && j
```

4.2 다음의 객체정의가 옳은가를 생각해 보시오.

```
bool P = false;
bool Q = true;
bool R = true;
int i = 1;
int j = 0;
```

이때 다음의 값을 평가하시오.

```
1) P && Q || !P && !Q
2) P || Q && !P || !Q
3) o == 1 || o < 1
4) o == 1 == true;
5) P && (Q || R)
6) !!P
7) !j
8) false < true
9) j = i
```

4.3 연산자수속원리를 리용하여 다음의 값을 괄호안에 넣으시오.

```
1) 1 + 2 == 3 * 4
2) 1 + 2 == 3 * 4
3) P == Q <= R
4) P < Q == R
5) P == Q || R
6) P == Q && R
7) !5 < 3 < 4
8) P || Q && R || 5
9) -5 < 9 == P && 3 == 17
```

련습 4.4부터 4.9까지는 i와 j를 int형으로 가정하시오.

4.4 i와 j가 같을 때 참인 식을 정의하시오.

4.5 6부터 9사이에 i가 놓일 때 참인 식을 정의하시오.

4.6 i가 짝수이고 j가 홀수일 때 혹은 i가 홀수이고 j가 짝수일 때 참인 식을 정의하시오.

4.7 다음의 조건을 모두 만족한다면 참인 식을 정의하시오. 즉 i가 11보다 더 크며 j가 28이상이며 m과 n은 서로 다른 값을 가진다.

4.8 다음 조건을 만족시키지 못하면 참인 식을 정의하시오. 즉 i와 j의 합이 30이고 i가 4보다 작으며 i와 j의 적은 54보다 더 크다.

4.9 다음의 조건을 만족한다면 참인 식을 정의하시오. 즉 i는 j의 값의 2배이며 j는 k보다 더 작지만 n보다는 더 크며 m은 부수이다.

4.10 다음의 코드토막을 분석하시오.

```
if (i <= j)
    cout << "1"<<endl;
else
    cout << "2"<<endl;

    cout << "3"<<endl;
```

이때 다음의 값을 평가하시오.

1) i가 1이고 j가 2이면 출력은 무엇인가?
2) i가 2이고 j가 1이면 출력은 무엇인가?
3) i가 2이고 j가 2이면 출력은 무엇인가?

4.11 다음의 코드토막을 분석하시오.

```
if (i == j)
    cout << "1"<< endl;
else if ((i % j) < 3)
    cout<< "2"<<endl;
else if (i < (j-1))
    cout << "3"<<endl;
else
    cout << "4"<<endl;
    cout << "5"<<endl;
```

이때 다음의 값을 평가하시오.

1) i가 9이고 j가 4이면 출력은 무엇인가?
2) i가 4이고 j가 9이면 출력은 무잇인가?
3) i가 5이고 j가 6이면 출력은 무엇인가?
4) i가 5이고 j가 9이면 출력은 무엇인가?

4.12 다음의 동작을 수행하는 코드토막을 작성하시오. j에 의하여 i를 나눈것이 4이면 i는 100으로 설정한다. i와 j의 적이 8이면 그때 i는 50으로 설정하며 j는 60으로 설정한다. 만일 i가 j보다 작다면 j는 2배로 되며 대신에 i가 우수이면 i는 2배로 된다. 다른 경우 i와 j는 1을 증가한다. 만일 i와 j가 0이면 i는 1로 하고 j는 2로 설정하며 대신에 i가 0이면 그때 i는 5로, j는 10으로 설정하며 대신에 j가 0이면 i는 10으로, j는 5로 설정하면 다른 경우 i와 j 를 둘다 4로 설정한다.

4.13 다음의 if명령문을 분석하시오.

```
if (( i == 3) || (j == 4)){
    cout << "yes"<<endl;
}
```

```
else {
    cout << "no"<<endl;
}
```

"yes"와 "no"가운데 표시되는 값이 있는가? 왜 그런가?

4.14 다음의 객체정의가 옳다고 가정하시오.

```
bool P = 거짓;
bool Q = 참;
bool R = 참;
```

이때 다음의 식을 분석하시오. 그것들이 간단히 평가되는가, 그렇다면 어디서인가를 결정하시오.

```
Q && P && R
Q && P || R
!Q || (i != j)
(P || Q && R) && (3 <= 4)
R || (P || !Q || !R && (4 > 3))
```

4.15 bool형객체 A, B, C, D를 리용하여 다음의 코드토막을 분석하시오.

```
if(A && B)
    if(!C || !D)
        cout << "1"<<endl;
    else if (D)
        cout<< "2"<<endl;
    else
        cout << "3"<<endl;
else if (C!=D)
    cout << "4"<<endl;
else if (c)
    cout << "5"<<endl;
else
    cout << "6"<<endl;
```

표준출력명령으로 1을 표시하기 위한 코드토막을 만드는 A, B, C, D값을 주시오.
표준출력명령으로 2를 표시하기 위한 코드토막을 만드는 A, B, C, D값을 주시오.
표준출력명령으로 3을 표시하기 위한 코드토막을 만드는 A, B, C, D값을 주시오.
표준출력명령으로 4를 표시하기 위한 코드토막을 만드는 A, B, C, D값을 주시오.
표준출력명령으로 5를 표시하기 위한 코드토막을 만드는 A, B, C, D값을 주시오.
표준출력명령으로 6을 표시하기 위한 코드토막을 만드는 A, B, C, D값을 주시오.

4.16 다음의 코드토막을 계산하는 진리값표를 작성하시오.

```
if (P)
```

```
        Operation = true;
    else if (Q)
        Operation = false;
    else
        Operation = true;
```

4.17 다음의 코드토막을 계산하는 진리값표를 작성하시오.

```
    if(P)
        if (Q)
            Operation =true;
        else
            Operation = false;
    else
            Operation =false;
```

4.18 같은 형태 (iso)의 론리2진연산을 위한 진리값표를 작성하시오. 연산은 연산수가 같은 값으로 평가되면 참이 되고 다른 경우 iso는 거짓으로 평가된다. iso를 계산하는 C++연산자가 있는가? 설명하시오

4.19 배타적론리합(xor)론리2진연산을 위한 진리값표를 작성하시오. 연산은 연산수중 하나가 정확히 참이면 참으로 평가되고 다른 경우 xor는 거짓으로 평가한다. P xor Q가 참인가를 검사하는 코드토막을 작성하는데 P와 Q는 bool형객체이다.

4.20 적의 부정(nand)을 위한 론리2진연산진리값표를 작성하시오. 연산은 두 연산수가 참이면 거짓이 되고 다른 경우 nand는 참으로 평가된다. P nand Q가 참인가를 검사하는 코드토막을 작성하는데 P와 Q는 bool형객체이다.

4.21 다음의 론리식을 위한 진리값표를 작성하시오.

1) (not P) and Q
2) P and ((not P) or Q)

4.22 드 모르간의 법칙은 론리행변수 P와 Q에 대하여

- not(P and Q)는 (not P) or (not Q)와 같다.
- not(P or Q)는 (not P) and (not Q)와 같다.

라는 능력을 표시한다. 진리값표의 사용을 통하여 2개의 동일성을 증명하시오.

풀이방향 : 첫 같기에서 진리값표는 P;Q;P and Q; not P; notQ; (int P) or(not Q)제목을 가진다.

4.23 year의 값이 윤년에 해당된다면 참인 옹근수형객체 year를 리용하여 론리식을 개발하시오.

4.24 다음의 프로그람흐름도를 작성하시오.

프로그람 4-2
프로그람 4-6
프로그람 4-7
프로그람 4-13

4.25 if-else-if명령문을 사용하여 switch명령문이 재실행될수 있다는것을 설명하시오.

4.26 여러개의 if-else-if명령문을 switch명령문으로 재실행할수 없다는것을 설명하시오.

4.27 입력으로 하나의 옹근수값을 얻는 프로그람을 실행하시오. 프로그람은 입력값이 정수인가 부수인가 혹은 0인가를 표시한다.

4.28 입력으로 2개의 공학값을 얻는 프로그람을 작성하시오. 프로그람은 둘 다 정수인가 둘 다 부수인가 혹은 하나는 정수이고 하나는 부수인가를 표시한다.

4.29 입력으로 2개의 류점수값을 얻는 프로그람을 실행하시오. 프로그람은 두 값의 차가 ε이상인가를 결정하는데 ε은 0.00001과 같이 프로그람이 정의한 상수이다.

4.30 가장 작은 값을 구하기 위하여 if-else-if명령으로 프로그람 4-2를 다시 작성하시오.

4.31 4개의 수를 배렬해야 한다. 서로 다른 배렬이 몇개 있는가?

4.32 4개의 입력값을 배렬하는 프로그람을 설계하고 작성하시오.

4.33 다음의 switch명령문을 분석하시오.

```
switch ( i * j ){
    case 1: case 2: case 3: case 6:
          cout << " 1 "<<endl  ;
    case 5:
          cout  <<  "2 "<< endl  ;
          break  ;
    case 10:
          cout  << "3 " << endl  ;
          break  ;
    default:
          cout  << " 4 " << endl  ;
}
```

이때 다음의 값을 평가하시오.

1) i가 11이고 j가 2이면 출력은 무엇인가?
2) i가 1이고 j가 5이면 출력은 무엇인가?
3) i가 3이고 j가 2이면 출력은 무엇인가?
4) i가 5이고 j가 7이면 출력은 무엇인가?

4.34 하나의 옹근수값을 얻는 프로그람을 작성하시오. 입력값은 1과 10사이의 범위에 있다. 프로그람은 입력값이 소수인가를 표시한다. switch명령문을 사용하여 입력값을 식별한다.

4.35 모드연산자 %가 들어 있는 <왼쪽연산수 % 오른쪽연산수>형식의 나머지식을 계산하기 위하여 프로그람 4-4를 수정하시오.

4.36 작기(<) 혹은 크기(>)연산자가 들어 있는 <왼쪽연산수 연산자 오른쪽연산수>형식의 간단한 관계식을 계산하기 위하여 프로그람 4-4를 수정하시오.

4.37 switch명령문으로 다음의 코드토막을 고치시오.

```
if (( i = = 1 ) || ( i = = 4 ){
    n = 1;
}
else if ( i = = 3 ) {
```

```
        n = 4 ;
    }
    else { n = 3 ;
    }
```

4.38 다음의 코드토막에서 잘못된것이 무엇인가? 정확히 어떻게 할수 있는가?

```
    cout << "Enter  key  Value(n):";
    int key;
    cin >> key;
    int  Value ;
    cout << "Enter a list of numbers(n1,n2,…):";
    for (int i = 0 ; cin >> Value ; ++i ) {
        if (key == Value){ ++counter;}
    }
    cout<<counter<<"of the"<<i<<"Values equal "<<key <<endl;
```

4.39 다음의 코드토막에서 잘못된것이 무엇인가? 정확히 어떻게 할수 있는가?

```
    int equal 1 = 0 ;
    int equal 2 = 0 ;
    int equal 3 = 0 ;
    cout << "Enter a list of numbers : (n1, n2, …) :";
    int  Value  ;
    while ( cin>> Value ) {
        switch (Value) {
            case 1: ++equal1;
            case 2: ++equal2;
            case 3: ++equal3;
         }
    }
    cout << equal 1 << "inputs  equals  1"<< endl ;
    cout << equal 2 << "inputs  equals  2"<< endl ;
    cout << equal 3 << "inputs  equals  3"<< endl ;
```

4.40 주간 날들에 대하여 enum형으로 정의하시오.

4.41 카드쌍을 위한 enum형을 정의하시오.

4.42 do명령문을 while명령으로 변화하는 방법을 말하시오.

4.43 c는 enum형색객체라고 하자. 표준출력지령 cout로 값의 이름을 넣는 switch명령문을 작성하시오. 실례를 들어 c가 값이 red를 가진다면 문자렬 "red"가 들어 간다.

4.44 프로그람 4-6에 입력 Validation을 추가하시오.

4.45 다음의 코드를 들여쓰기하시오.

```
if(( n > 0) && (m > 0))
     { for (int  i = 0; i < n; ++i )
     { for ( int  j = 0; j < m; ++j ){
     if(i! =j {cout<<"0"<<endl;}else{ cout<<"1";
     }}}} else {cout << " 2 "<< endl ;}
```

4.46 0부터 99사이에서 얻은 값을 그대로 출력하는 코드를 작성하시오. 그때 토막은 단어들에서 값을 표시한다. 실례를 들어 입력값이 21이면 21이 표시된다.

4.47 다음의 코드토막의 출력은 무엇인가?

```
int counter1 = 0;
int counter2 = 0;
int counter3 = 0;
int counter4 = 0;
int counter5 = 0;
for (int i = 0 ; i < 10 ; ++i) {
    ++counter1 ;
    for(int j = 0; j < 10; ++j){
        ++counter2;
        if ( i == j ) {
            ++counter3 ;
        }
        else {
            ++ counter4 ;
        }
    }
    ++counter  5;
}
cout << counter1 << " "<< counter2 <<" "
     << counter3 <<" "<< counter4 <<" "<< counter5;
```

4.48 아래의 코드를 수정하여 입력하는 수가 먼저 입력한 수보다 더 클 때 출력하도록 하시오.

```
int FirstValue ;
int CurrentValue ;
int Sum = 1 ;
cin >> FirstValue ;
while (cin >> FirstValue ) {
    if(FirstValue == CurrentValue){
        ++Sum ;
    }
}
```

```
cout << Sum << endl;
```

4.49 아래의 코드를 수정하여 1부터 n까지의 홀수들의 합을 표시하도록 하시오.

```
int i = 0 ;
for (int Sum = 1 ; Sum < n ; ++i){
    if (i % 2) {
        Sum += n;
    }
    cout << Sum << endl ;
}
```

4.50 사용자로부터 4각형의 특징값들인 너비와 높이, 중심자리표 (x, y), 색을 입력하여 W창문에 RectangleShape를 리용하여 4각형을 그리는 코드를 작성하시오. 4각형의 치수와 중심자리표는 류점수형이며 4각형의 색은 문자렬이다. 실례로 사용자로부터 값을 입력하는 형식은 아래와 같다.

```
4각형의 지표(w h x y color)를 입력하시오 : 4 5 2 2 red
```

4.51 for명령문을 사용하여 11부터 29사이의 옹근수의 합을 표시하는 코드를 작성하시오.

4.52 4옹근수의 n쌍을 련속 얻는 코드토막을 주시오. 옹근수 a와 b의 매 추출된 쌍을 위한 프로그람결과는 a*a+1* … b를 표시한다.

4.53 입력한 수의 절대값을 표시하는 프로그람을 작성하시오.

4.54 다음의 코드를 수정하여 5와 15사이의 옹근수를 표시하도록 하시오.

```
int  Factor = 5 ;
int  product = 1 ;
    do {
        ++ Factor ;
        Product * = Factor ;
} until (Factor == 15) ;
cout << Product << endl ;
```

4.55 다음의 코드를 분석하시오.

```
int i = 1;
while(i <= n){
    if((i % n ) == 0){
       ++ i;
    }
}
cout << i << endl;
```

이때 다음의 값을 평가하시오.

n이 0이면 출력은 무엇인가?

n이 1이면 출력은 무엇인가?

n이 3이면 출력은 무엇인가?

4.56 다음의 코드토막을 분석하시오.

```
for(i = 0; i < n ; ++i){
--n ;
}
cout << i << endl;
```

이때 다음의 값을 평가하시오.

n이 0이면 출력은 무엇인가?
n이 1이면 출력은 무엇인가?
n이 3이면 출력은 무엇인가?
n이 4이면 출력은 무엇인가?

4.57 입력한 값들중에서 정수와 부수 그리고 0의 개수를 계산하는 코드를 작성하시오.

4.58 표준입력흐름에서 값을 읽어 표준출력흐름으로 최소값과 최대값을 표시하는 프로그람을 작성하시오. 프로그람은 입력이 없고 한번에 입력하는 부분에 적당한 통보를 표시한다.

4.59 사용자로부터 부수가 아닌 값 n을 입력하여 BCD코드형식으로 출력하는 프로그람을 작성하시오. 실례로 입력값이 19이면 출력은 11001이다.

4.60 사용자로부터 부수가 아닌 값 n을 입력하여 표준적인 2진표기법으로 표시하는 프로그람을 작성하시오. 실례로 입력한 값이 21이면 표시되는 값은 10101이다.

4.61 식의 종류를 얻고 매식의 출력을 표시하도록 프로그람 4-4를 수정하시오.

4.62 날자를 입력하여 그해 정초부터 그날까지의 날자수를 계산하는 프로그람을 작성하시오. 실례로 "2002년 12월 29일"이라고 입력하면 "363일"이 출력된다.

4.63 프로그람 4-5를 수정하여 시작날자와 마감날자를 입력하여 그사이의 날자수를 계산하는 프로그람을 작성하시오.

4.64 보통 수렬은 일정한 규칙으로 정렬되여 있다. 실례로 수렬 {1, 1, 3, 4, 9}는 커지는 순서로 배렬되여 있지만 수렬 {1, 3, 2, 4, 9}는 3이 2보다 크므로 정렬되지 않았다. 임의의 수렬을 입력하여 그 수렬이 정렬되여 있는가 아닌가를 검사하는 프로그람을 작성하시오.

4.65 사용자로부터 부수가 아닌 값 n을 입력하여 아래와 같은 형식으로 출력을 진행하는 프로그람을 작성하시오..

```
1    2    3    …    n-1    n
1    2    3    …    n-1
…    …    …
1    2    3
1    2
1
```

4.66 입력하는 문자렬에서 공백(" "과 "\ t"(타브건에 해당))의 개수를 세는 프로그람을 작성하시오.

4.67 입력한 문자렬의 개수와 문자렬들의 평균길이를 구하는 프로그람을 작성하시오.

4.68 목록 4-2의 문자처리방법을 리용하여 문자의 중복을 감시하도록 프로그람 4-9를 다시 작성하시오.

제 5 장. 함수

소 개

함수를 리용하면 프로그람이 명백해 지며 쏘프트웨어를 재리용할수 있다. 함수는 특정한 과제를 수행하여 그 풀이를 주는 방조자와 같다. 다음장들에서 함수의 설계와 리용방법을 구체적으로 취급한다. 이 장에서는 함수의 호출과 파라메터넘기기 등 기초적인 개념들을 취급한다. 이를 위하여 iostream서고를 비롯한 표준쏘프트웨어서고함수들을 리용하여 프로그람을 작성한다. iostream 서고는 객체를 현시하고 추출하는 확장가능한 방법들을 제공하므로 중요하다. 서고는 전처리기지령을 리용하여 참고할수 있다. 전처리기지령은 파일포함, 마크로정의, 조건부콤파일을 지원한다.

기본개념

- 함수
- 값파라메터
- 호출과 조종흐름
- 머리부파일
- 함수원형선언
- 활성화레코드
- define지령
- 파일포함
- 조건부콤파일
- iostream기능
- 모조란수
- iomanip조작자
- 형식화된 출력
- fstream클라스 ifstream
- fstream클라스 ofstream
- 파일조작
- stdlib서고
- exit()함수
- assert서고
- 번역단위
- 형변환

5.1 함수의 기초

앞장에서 나오는 프로그람들은 흥미는 있지만 그것은 표준쏘프트웨어가 아니다. 이 프로그람들에서는 iostream서고와 EzWindows도형서고를 제쳐 놓으면 그 프로그람안에서의 동작은 개별적인 함수 main()과 ApiMain()안에서 완전히 규정된다. 이것은 과제가 단순하고 코드작성이 쉬웠으므로 가능하였다. 그러나 규모가 큰 쏘프트웨어응용프로그람은 비록 수백만까지는 안된다 하더라도 수십만개의 코드행을 요구한다. 이런 거대한 응용프로그람에 대한 단일한 코드토막을 정확히 작성한다는것은 실지로 불가능하며 이런 크기의 거대한 하나의 프로그람은 너무 복잡하여 리해할수도 검사할수도 없다. 응용프로그람이 이런 식으로 작성될수 있다 해도 그것은 자원랑비이다. 그것은 쉽게 리용할수 있는 프로그람서고에서 일반계산과제를 수행하는 많은 프로그람모듈이 이미 있기때문이다. 이러한 리유로 하여 모든 중요한 응용프로그람개발과 지어는 가장 단순한 프로그람들도 모듈형식으로 정보와 그 조작방법을 조직하는 프로그람작성도식을 리용한다. 모든 그러한 도식의 주되는 부분이 함수를 리용하는것이다. 함수는 객체지향프로그람을 작성하는 중요한 요소이다.

함수는 프로그람들이 특정한 과제를 풀수 있게 하여 준다. C++에서 프로그람은 다음명령을 계속하기에 앞서 함수가 자기 과제를 완성할 때까지 대기한다. 함수는 과제수행을 위하여 다른 함수를 리용할수 있다. 사실상 함수는 지어 자기의 또 다른 구체례도 리용할수 있다(이 처리를 재귀처리라고 하는데 다음장에서 취급한다).

프로그람 5-1은 사용자가 입력한 값들을 결수로 하는 2차방정식의 뿌리를 함수 main()이 계산하도록 하는 sqrt()함수를 리용하는 간단한 프로그람이다. a, b, c를 결수로 하는 2차방정식은

$$ax^2 + bX + c = 0$$

이다. 방정식의 풀이는 다음의 공식

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

으로 계산할수 있다. 교육의 목적에 따라 프로그람 5-1은 2개의 실수풀이를 가지는 2차방정식에 대한 계산으로 제한한다(일반2차방정식에 대한 문제는 련습에서 계산한다).

함수를 리용하는 동작을 호출(invocation, call)이라고 한다. 프로그람 5-1에서 함수 sqrt()는 double형객체 radical의 정의에서 호출된다.

```
double redical = sqrt(b*b-4*a*c);
```

함수는 호출될 때 과제수행을 위한 정보를 넘겨 받을수 있다. 이 정보를 함수의 파라메터 혹은 인수라고 한다.

```
// 프로그람 5-1: 2차방정식의 뿌리계산
#include <iostream>
#include <string>
#include <cmath>
using namespace std;
int main(){
    cout << "결수들을 입력하시오.: ";
    double a;
    double b;
    double c;
    cin >> a >> b >> c;
    if ((a!=0) && ((b*b-4*a*c)>0))  {
        double radical = sqrt(b*b-4*a*c);
        double root1 = (-b+radical)/(2*a);
        double root2 = (-b-radical)/(2*a);
        cout << "The root s of"0< a << "0**2+•0 << b << "0+"0 << c << "0re
          << root1 << "0nd•0<root2 << end1;
    }
    else {
```

```
        cout << a << "0**2 +·0<< b << "0+"0<< c << "Does not have two real
            roots·0 << endl;
    }
    return 0;
}
```

프로그람 5-1. 2차방정식 ax^2+bx+c의 뿌리계산

실례로 함수 sqrt()는 파라메터로서 double형의 류점수값을 넘겨 받을것을 예견하며 다음 그 파라메터의 2차뿌리를 계산한다.

함수가 호출될 때 조종흐름(flow of control)이 그 함수에로 림시 넘어 가는데 이것은 다음 실행될 명령문이 호출된 함수의 첫 명령문이라는것을 의미한다. 함수정의에 대한 실행의 준비로 호출에 리용된 실제값과 함수정의에 있는 파라메터의 대응관계가 설정된다. 호출된 함수가 과제를 완성한후에 조종흐름은 함수를 호출한 명령문으로 돌아 간다.

5.1.1 대면부명세

프로그람을 번역할 때 함수에 대하여 3가지 사항 즉 되돌려야 할 값의 자료형과 함수의 이름, 그리고 그의 파라메터를 명백히 규정하여야 한다. 이 3가지 항목들은 함수의 대면부(function’s interface)를 이룬다. 실제적인 함수정의는 요구되지 않는다(비록 콤파일처리완성을 위하여 결국 정의는 되겠지만)는 점에 주의하여야 한다.

앞의 실례프로그람들에서는 main()의 여러가지 정의들에서 함수대면부의 실례들을 보았다. 대면부는 왼쪽대괄호의 앞에 있는 부분이다. 그러므로 프로그람들에서 함수 main()의 대면부는

```
int main()
```

이다. 프로그람 5-1에서 리용한 함수 sqrt()의 대면부는 math서고머리부파일 cmath에 지정되여 있는데 머리부파일은 함수대면부들과 상수, 변수, 객체정의 그리고 클라스서술들의 어떤 집합이다. 머리부파일은 include지령을 리용하여 프로그람에 포함시킬수 있다.

```
#include <cmath>
```

cmath에서 c는 math서고가 C언어에서 받아 들였다는것을 가리킨다. math머리부파일에 있는 sqrt()함수의 대면부명령문은

```
double sqrt(double number);
```

이다. 대면부명세(interface specification)의 첫 부분은 함수가 어떤 형의 값(만일 있다면)을 만들어 내는가를 가리킨다. 이것을 함수형(function type) 또는 복귀형(return type)이라고 부른다.

함수형은 표준자료형일수도 있고 프로그람작성자가 정의한 자료형일수도 있다. sqrt()함수의 형은 double이다.

함수가 값을 얻어 내면 이 값을 되돌림값이라고 한다. 함수의 형이 void가 아니면 함수는 언제나 규정된 자료형을 가진 한개의 값만을 되돌린다.

함수에서 얻어 진 되돌림값은 함수를 호출할 때마다 달라 진다. 실례로 sqrt()함수가 파라메터값을 6.25로 넘겨 받으면 이때 되돌림값은 2.5로 된다. 만일 파라메터값이 1.44이면 되돌림값은 1.2이다.

함수형이 void이면 함수는 되돌림값이 없다. 되돌림값이 없는 함수가 있다는것이 이상하게 들릴지

모르지만 void함수들은 매우 편리할수 있다. 실례로 이런 함수들은 사용자에게 통보를 현시하는데 자주 리용된다.

대면부명세의 다음부분은 함수이름이다. 이름은 반드시 식별자로 되여야 한다. 프로그람작성자는 보통 객체이름과 같이 함수이름에도 같은 이름붙이기약속을 적용하는데 이것은 일관한 대문자사용도식을 가진 좋은 이름을 리용한다는것을 의미한다. 이름붙이기약속에서는 특별한 조건이 없는한 함수이름에서 매 단어의 첫 문자는 대문자로 한다. 이런 도식은 독자들이 함수가 프로그람작성자에 의해 정의된것인가 표준서고에 있는것인가를 알기 쉽게 해준다(거의 모든 표준서고함수들의 이름은 소문자로 이루어 진다).

C++언어

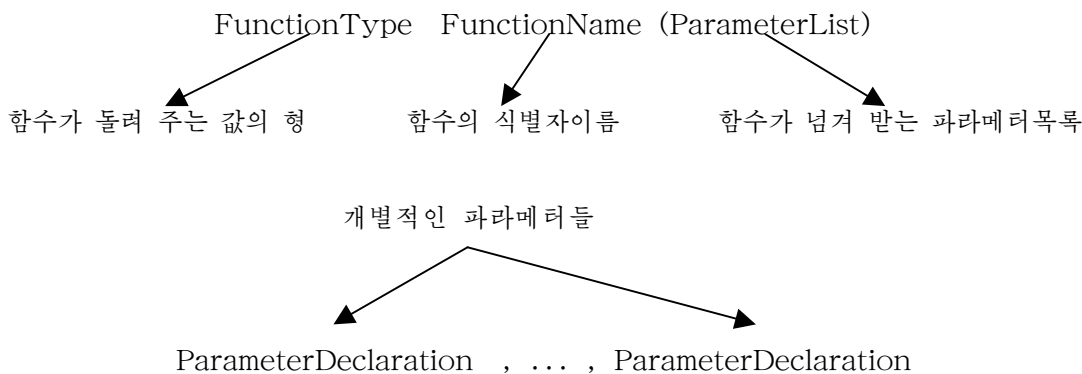
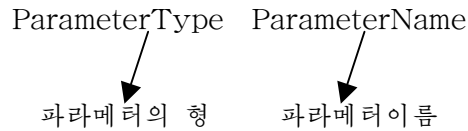
C에 있거나 없는것

이전의 C++콤파일러는 표준서고들의 표준이름짓기규정을 지원할수 없다. 사용자의 콤파일러가 머리부파일이름에서 앞붙이 C가 없이 math서고나 다른 C서고들을 호출할 필요가 있다는것을 알아야 한다. 또한 일부 콤파일러들은 파일확장자 .h를 어떤 표준머리부파일에 리용할것을 요구한다. 실례로 일부 콤파일러들은 math서고가 다음의 방식으로 포함될것을 요구한다.

```
#include <math.h>
```

서고를 어떻게 포함시키는것이 그것이 명시하는 객체, 형, 함수의 참조에 어떤 영향을 줄수 있는가에 주의하여야 한다. h뒤붙이가 없는 머리부파일들은 보통 이름공간에 그 원소들을 배치한다. 그러한 머리부파일원소들을 리용하기 위하여 using name space 명령문을 리용한다. h뒤붙이가 붙은 머리부파일들은 using namespace명령을 요구하지 않는 이름공간에 자기의 원소들을 배치한다. 이름공간에 대한 론의는 부록 4를 보면 된다.

대면부명세의 마지막부분은 함수에 넘기는 파라메터들의 형식을 서술한다. 파라메터들은 괄호 ()안에 서술된다. 이미 본 프로그람실례들에서 main()도 ApiMain()도 표준입력흐름을 통하여 쓰일수 있다는것외에는 그 어떤 정보도 요구하지 않았다. 따라서 그것들의 파라메터목록은 비여 있다. 그러나 sqrt()의 경우는 다르다. 이 함수는 계산에 필요한 하나의 정보토막 즉 2차뿌리를 계산하는데 필요한 값을 요구한다. 이러한 정보토막의 서술과정을 파라메터선언(parameter declaration)이라고 한다. 여러개의 정보토막을 요구하는 함수는 대응되는 여러개의 파라메터로 선언되는데 매개 선언은 반점으로 구분해 주어야 한다.

함수대면부명세는 다음과 같은 형식을 가진다.

C++는 프로그람작성자들에게 파라메터지정을 위한 확장된 선택항목모임들을 제공한다. 현재는 값파라메터선언형식만을 고찰하자. 다음장에서 다른 형식들을 고찰한다. 기본형식에 있어서 값파라메터선언은 객체선언과 같이 파라메터형이 있고 그다음에 식별자가 놓인다.

sqrt()함수는 이 파라메터선언의 기본형태를 리용하는데 그 파라메터선언은 다음과 같다.

```
double number
```

5.1.2 함수원형선언

함수원형명령문은 지정된 형식의 함수가 프로그람의 부분과제를 수행하는데 리용될수 있다는것을 가리킨다. 함수원형(function prototype)명령문은 대면부명세와 비슷한데 그뒤에 반두점이 놓인다. 앞에서 언급한것처럼 대면부는 적어도 함수가 프로그람에서 리용되기전에 지정되여야 한다. 함수원형명령문은 #include명령문다음의 프로그람파일의 시작점가까이에 배치하는것이 일반적이다.

간단한 원형선언의 실례를 아래에 주었다.

```
int PromptAndExtract();
float CircleArea(float radius);
bool Isvowel(char CurrentCharacter);
```

PromptAndExtract()원형은 그 함수가 파라메터를 예견하지 않으며 int형값을 돌려 준다는것을 가리킨다. CircleArea()원형은 그 함수가 float형의 값을 계산하여 돌려 준다는것을 가리킨다. 또한 CircleArea()는 그것에 넘겨 지거나 주어 지는 프로그람작성자용어에서 하나의 float값을 요구한다. Isvowel()에 대한 원형은 bool값을 되돌리며 함수는 char형값을 넘겨 받을것을 예견한다.

함수정의를 완성하려면 대면부앞에 함수의 작용을 가리키는 명령문목록(함수본체)을 배치하여야 한다. 명령문목록은 왼쪽과 오른쪽대괄호안에 들어 있다. 프로그람 5-1에 있는 main()함수의 경우를 보시오.

우에서 함수원형이 대면부명세와 비슷하다고 하였는데 그것은 파라메터들의 이름이 원형에 반드시 필요한것이 아니기때문이다. 그러나 프로그람의 리해를 도모하기 위하여 좋은 이름을 포함하기로 한다.

5.1.3 호출과 조종흐름

프로그람 5-1에서 radical의 정의를 보면 초기화부문에서 함수 sqrt()를 호출한다.

```
double radical= sqrt(b*b - 4*a*c)
```

함수를 호출할 때 2차뿌리를 구하려는 값을 제공하는데 우의 경우에는 그 값이 식 b*b-4*a*c의 값이다. 이때 식 b*b-4*a*c를 실제파라메터라고 한다. 실제파라메터를 표현하기 위하여 sqrt()정의에서 리용된 객체들을 형식파라메터라고 한다. 프로그람을 실행할 때 이 점에서 조종흐름이 림시 main()에서 sqrt()로 넘어 간다. 그림 5-1에 프로그람 5-1의 조종흐름을 주었다.

sqrt()함수에로 조종을 넘길 때 호출에서 주어 진 실제파라메터와 그 파라메터를 표현하기 위하여 sqrt()에서 리용된 객체사이의 대응을 설정하여야 한다. 일반적으로 파라메터들의 상대위치는 실제파라메터와 형식파라메터사이 대응관계를 결정한다. 첫번째 실제파라메터는 첫번째 형식파라메터에, 두번째 실제파라메터는 두번째 형식파라메터에 관련된다. 함수의 대면부가 기초적인 값파라메터선언형식을 리용한다면 함수를 호출할 때마다 모두 같은 수의 파라메터를 가져야 한다.

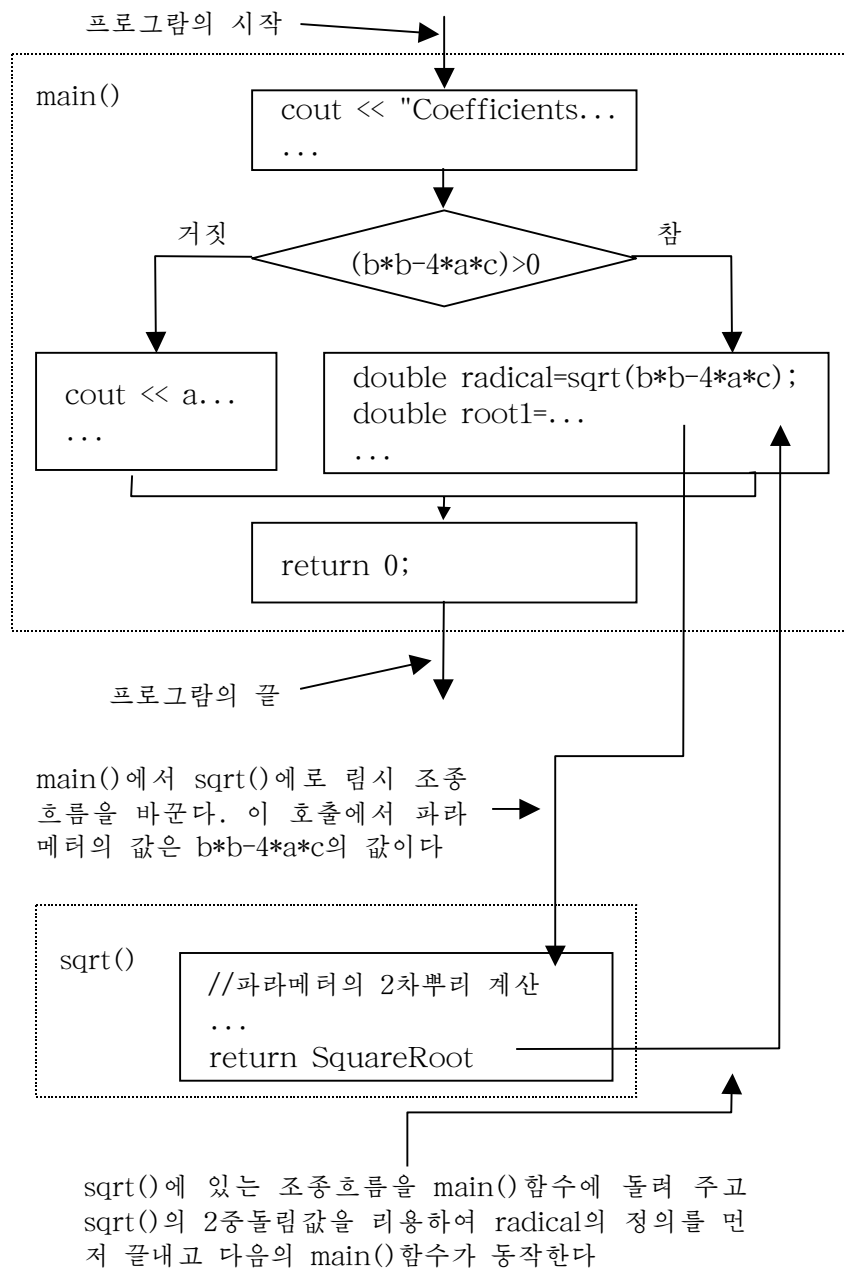



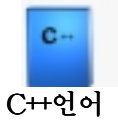
그림 5-1. main()에서 sqrt()에로, 다시 main()에로 진행되는 프로그람 5-1의 조종흐름

함수를 호출할 때마다 형식파라메터에 기억기가 따로 할당된다. 값형식파라메터를 가지는 sqrt()와 같은 함수에 대하여 형식파라메터의 기억기는 실제파라메터값으로 초기화된다.

함수의 형식파라메터에 따로 할당된 기억기는 함수의 활성화레코드(activation record)의 한 부분이다. 새로운 활성화레코드는 함수를 호출할 때마다 창조된다. main()함수도(조작체계도 호출된다.) 활성화레코드를 가진다.

함수의 활성화레코드는 함수에서 정의된 모든 객체와 관련된 값을 기억할수 있는 크기를 가진다. 콤파일러에 따라 다른 정보가 활성화레코드에 남아 있을수 있다(실례로 실행되고 있는 함수의 현재 명령문에 대한 지적자, 실행되고 있는 함수를 호출한 명령문에 대한 지적자, 되돌림값이 있는 경우 그 값을 림시 기억하고 있는 기억기 등).

프로그람 5-1에서 main()함수의 활성화레코드에는 a, b, cl radical, ro ot1, root2가 있다. 사용자가 a, b, c에 준 값이 각각 6, 5, 1이고 main()함수에서 radical의 초기화명령을 실행하려고 한다고 가정하자.

C++언어

자료형일치

함수호출에 리용된 실제파라메터의 자료형은 함수대면부에 지정된 대응하는 형식파라메터의 자료형과 같아야 한다. 실제파라메터와 형식파라메터사이에 차이가 있다면 일반적인 변환이 자동적으로 집행된다. 일반변환이 진행될 때 실제파라메터를 대응하는 형식파라메터의 형으로 변환할 수 없다면 콤파일오유가 생긴다.

main()함수의 활성화레코드에 대한 다음의 그림에서 물음표는 객체의 값이 프로그람에서 아직 설정되지 않았다는것을 지적한다.

Main()	
a	6
b	5
c	1
radical	?
root1	?
root2	?

sqrt()함수가 radical초기화를 위해 호출될 때 sqrt()함수의 활성화레코드가 얻어 진다. 그 활성화레코드에서 값형식파라메터는 실제파라메터 즉 식 b*b-4*a*c의 값으로 초기화되는데 이 경우는 그 값은 1이다.

b*b-4*a*c가 형식파라메터의 초기화에 일단 리용되였다면 실제파라메터와 형식파라메터는 서로 독립이다. 형식파라메터가 sqrt()에 리용될 때마다 형식파라메터에 리용된 값은 sqrt()의 현재활성화레코드에 기억된것이다. sqrt()함수의 호출이 형식파라메터를 변화시킬지라도 그 호출활성화레코드의 기억기안에서 변화가 생기며 식 b*b-4*a*c는 영향을 받지 않는다. 형식파라메터와 실제파라메터사이의 이러한 형태의 관계를 값에 의한 넘기기(pass by value)라고 한다. 본질에 있어서 형식파라메터는 sqrt()함수안에서만 리용될수 있는 하나의 객체이다. 형식파라메터는 sqrt()호출과 함께 창조되며 sqrt()가 과제를 완성하고 활성화레코드의 기억기를 해방할 때 파괴된다.

다음의 코드토막은 sqrt()호출의 몇가지 합법적호출을 보여 준다.

```
double ScaledRoot = 25 * sqrt(1000);
sqrt(15, 0);
```

첫번째 호출은 scaledRoot를 초기화한다. 이 호출은 함수호출이 연산자와 조합하여 리용할수 있다는것을 보여 준다. 이 함수호출은 그것이 되돌림값을 가지며 그 되돌림값이 임의의 다른 값으로 리용될수 있기때문에 가능하다. 특히 되돌림값은 식을 구성하는데도 리용된다. 두번째 호출은 되돌림값으로 아무것도 하지 않으므로 이상해 보인다. 그러나 명령문은 틀리지 않는다. 이 식은 C++에서 정확한 명령문이다. 그래서 정확한 함수호출은 정확한 식이다.

다음의 명령을 고찰해 보자.

```
cout << sqrt(14) - sqrt(12);
```

이 삽입명령문은 sqrt()함수를 두번 호출한다. 따라서 조종흐름이 sqrt()함수에 두번 넘어 가며 2개의 sqrt()활성화레코드가 이 명령을 실행하는 동안 창조된다. 2개의 활성화레코드는 서로 다르며 하나가 생긴 후에 다음것이 생긴다. 즉 그것들은 동시적으로 존재하지 않는다. 함수를 호출할 때마다 활성화레코드의 기억구역(가능하면 매번 다른 기억위치를 할당)에 sqrt()형식파라메터값이 들어 간다. 첫번째 호출시 기억기는 값 14로 초기화되며 두번째 경우는 값 12로 초기화된다.

실제파라메터를 14로 호출할 때의 되돌림값은 덜기연산의 왼쪽연산수로, 실제파라메터를 12로 호출할 때의 되돌림값은 덜기연산의 오른쪽연산수로 리용된다.

개념적으로 다음의 코드토막은 sqrt(14)와 sqrt(12)의 삽입을 표현할수 있다.

```
double LeftOperand = sqrt(14);
double RightOperand = sqrt(12);
cout << LeftOperand - RightOperand;
```

이 명령문들이 덜기연산기의 왼쪽연산수를 먼저 평가하고 다음 오른쪽연산수를 평가한다 해도 원래의 삽입명령은 이런 평가순서화를 요구하지 않았다. C++우선권규칙에서는 론리식의 단락평가를 제외하고 연산자평가순위가 정확히 지정되여 있으나 연산자의 어느 연산수가 먼저 평가되는가는 지정하지 않고 콤파일러에 맡겨 둔다. 따라서 sqrt(12)을 호출한 다음에 sqrt(14)를 호출할수도 있다. 즉 다음의 코드토막 역시 sqrt(14)-sqrt(12)의 삽입을 나타낼수 있다.

```
double RightOperand = sqrt(12);
double LeftOperand = sqrt(14);
cout << LeftOperand - RightOperand;
```

주의

함수의 부작용

sqrt(14)- sqrt(12)를 평가할 때 두 연산수중 아무 연산수나 먼저 평가할수 있다. 그러나 다음의 장들에서 보면 식의 값이 평가순서에 따르는 함수를 작성할수 있다. 이 평가순서는 함수가 함수밖에서 정의된 객체를 수정할 때 생긴다. 이러한 수정을 그것이 함수의 되돌림값에 추가로 생긴다는데로부터 부작용(side effect)이라고 한다. 리해를 쉽게 하기 위하여 부작용을 내는 함수호출은 긴 식의 부분으로 되지 말아야 한다.

한개의 명령문에서 되돌림값을 연산수로 리용하는데 제한이 없이 다중호출을 할수 있다. 되돌림값은 또한 다른 함수호출의 실제파라메터로 리용될수 있다. 그러한 실례를 아래에 주었다.

```
double QuarticRoot = sqrt(sqrt(5));
```

이 실례에서 sqrt()함수는 두번 호출되고 서로 다른 활성화레코드가 결과적으로 창조된다. 그러나 이때 호출순서가 규정된다. Sqrt(5)식이 먼저 계산되고 다음 그 되돌림값을 두번째 호출에 리용한다. 이 호출순서는 sqrt()활성화레코드가 먼저 값 5로 초기화되는 형식파라메터에 대한 기억구역으로 창조한다는것을 의미한다. 다음에 함수는 이 활성화레코드를 리용하여 계산을 진행하고 sqrt()호출의 되돌림값을 얻는다. 다음 활성화레코드는 파괴되고 새로운 활성화레코드가 창조되며 sqrt()호출의 형식파라메터를 되돌림값으로 초기화한다. 이 값을 리용하여 함수는 다시 계산을 진행하고 QuarticRoot를 초기화하는데 리용되는 되돌림값을 생성한다. 따라서 다음의 작용이 발생한다(개념적으로).

```
double Temporary = sqrt(5);
```

```
double QuarticRoot = sqrt(Temporary);
```

모든 함수호출이 다 유효한것은 아니다. 실례로 다음의 2개 호출은 유효하지 않다. 첫번째 호출은 파라메터의 개수가 너무 작고 두번째는 너무 많다.

```
double x = sqrt();         //오유
double y = sqrt(5, 3) ;    //오유
```

더욱 유용한 C++서고에 대하여 구체적으로 론의하기전에 전처리를 고찰하고 서고머리부파일의 원형과 정의를 프로그람파일로 병합하자.

5.2 전처리기

여기서는 파일포함지령과 조건부콤파일지령에 대한 입력프로그람파일을 조사한다. 전처리기(preprocessor)가 입력프로그람파일에 대하여 이러한 지령을 수행하여 콤파일되여야 할 실제파일을 만들기때문에 이런 지령들의 처리는 프로그람번역의 첫 걸음으로 된다. 전처리기에 의하여 만들어 진 파일을 번역단위(translation unit)라고 한다.

5.2.1 파일포함지령

파일포함지령은 번역단위의 일부로 될 파일이름을 지정한다. 그 지령에서 이름 붙혀 진 파일은 지령그자체를 대신한다. 포함되여야 할 파일을 표준등록부에서 찾을수 없다면 번역처리가 중지되고 오유통보가 생긴다.

파일포함지령형식은 2가지이다. 한가지 형식을 아래에 주었다.

```
#include <파일이름>
```

#기호는 뒤따르는 전처리기지령을 가리킨다. #는 행의 첫 머리에 있어야 한다. 각괄호 <와 >기호로 파일이름을 경계 짓고 체계의 표준등록부에서 파일을 찾는다는것을 가리킨다. 이 표준등록부들의 위치는 체계마다 다를수 있다. 개인용콤퓨터체계에서 표준등록부들은 콤파일러가 있는 등록부의 부분등록부에 위치하는것이 보통이다. 이러한 서고들은 보통 콤파일러가 개인용콤퓨터상에 설치될 때 창조된다.

번역단위의 크기는 입력프로그람파일보다 훨씬 크다. 실례로 이 콤파일러를 리용하는 경우 다음의 5행짜리 프로그람파일이 884행짜리 번역단위를 가진다. 크기가 이렇게 늘어 난것은 iostream서고가 다른 표준흐름삽입과 추출과는 다른 더 많은 능력을 제공한다는것을 의미한다.

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    cout << "Hello, World" << endl;
    return 0;
}
```

파일포함지령의 두번째 형식은 다음과 같다.

```
#include "파일이름"
```

파일이름을 둘러 막고 있는 인용부호는 등록부들의 어느 한 모임을 탐색하여 문제의 파일을 찾아야 한다는것을 의미한다. 이 등록부의 고찰과정은 체계실현에 의존한다. 실제로는 거의 모든 콤파일러들이 다음과 같은 방식으로 동작한다. 만일 파일이름이 절대파일체계경로이름으로 주어 지면 그 파일을 절대위치에서 얻는다.

실례로 다음의 지령은 Sample.h를 포함시킨다는것을 말하는데 이 파일은 개인용콤퓨터체계의 C구동기의 \example\source등록부에서 찾을수 있다.

```
#include "C:\example\source\ sample.h"
```

절대경로이름이 지정되지 않으면 파일이름은 사용자가 현재 작업하고 있는 등록부에 대한 상대이름이라고 생각한다. 실례로 다음의 지령은 hold.txt파일의 현재등록부에서 보게 될것이다. 파일이 그 등록부에 없다면 표준등록부에서 탐색한다.

```
#include "hold.txt"
```

포함된 파일들이 전처리기지령을 포함하도록 하는것은 합법칙적이다. 파일이 지령을 포함한다면 그 지령들 역시 처리된다. 포함된 파일에 있는 지령의 처리는 현재파일에 있는 나머지지령들을 처리하기전에 진행된다. 실례로 다음의 행은 전처리에 입력된 파일의 내용을 나타낸다고 가정하자.

```
#include "a.txt"
```

a.txt파일은 다음과 같은 행을 포함한다.

```
#include "b.txt"
#include "b.txt"
```

그리고 파일 b.txt는 다음과 같은 행을 포함한다.

```
Hello world
```

전처리기의 출력은 다음의 행을 가진 파일이다.

```
Hello world
Hello world
```

창조되고 있는 행이 합법적명령문인가 아닌가를 고찰하지 않는다는것을 주의하여야 한다.

5.2.2 조건부콤파일

지금까지 프로그람에서는 모든 명령문들이 언제나 콤파일되였다. 이런한 특성은 #ifndef(만일 정의되지 않았다면)조건부콤파일지령을 가지고 재정의할수도 있고 ifdef(정의되였다면)를 가지고 무시할수도 있다.

전처리기지령 #ifndef는 파일에서 한개 부분의 열기구분문자(opening delimiter)로서 쓰인다. 전처리기지령 #endif는 그 부분의 닫기구분기호(closing delimiter)로서 쓰인다. 구분된 부분은 #ifndef다음에 오는 마크로이름이 사전에 정의되지 않았을 때에만 콤파일된다. 마크로이름은 #define명령문을 리용하여 정의된다.

실례로 전처리명령문

```
#define I386
```

은 이름 I386을 정의한다. 어느 명령문이 콤파일되여야 하는가를 결정하는 능력은 다중포함에 의하여 생기는 객체의 잘못된 재정의를 막을수 있게 하므로 쓸모가 있다. 실례로 많은 프로그람들이 iomanip서고가 iostream서고를 포함하는 경우에도

```
#include <iostream>
#include <iomanip>
```

로부터 시작하고 있다. iostream서고에 의하여 #ifndef명령문을 사용하지 않으면 cin이나 cout와 같은 표준흐름의 정의가 반복되므로 그런 프로그람은 반칙이라고 할수 있다. 다음의 실례에서 객체 a와 b는 이 코드토막이 프로그람파일에 여러번 포함되여도 한번만 정의된다.

```
#ifndef NO_MORE_A_AND_B
int a;
int b;
#define NO_MORE_A_AND_B
#endif
```

이 동작은 이 조건부콤파일지령안의 한행이 마크로 NO_MORE_A_AND_B를 정의하는 경우이다. 마크로 NO_MORE_A_AND_B가 정의되면 코드토막을 다시 포함시켜도 a와 b는 재정의되지 않는다.

조건부콤파일지령은 또 다른 조건부콤파일지령안에 매몰될수도 있다(이러한 매몰처리를 겹싸기(nesting)라고 한다). 실례로 다음의 코드토막에서 객체 a와 c는 마크로 NO_INTS가 정의되지 않을 때 정의된다. 객체 b는 마크로 NO_INTS와 NO_B가 정의되지 않았을 때만 정의된다. 조건부콤파일지령들이 겹싸기되면 #endif명령문은 가장 최근에 열린 #if명령문에 대응된다.

```
#ifndef NO_INTS
int a;
#ifndef NO_B
int b;
#endif
int c;
#endif
```

#ifdef지령도 비슷한 방법으로 동작한다. 그러나 련관된 명령문이 콤파일처리의 일부로 되도록 마크로이름이 정의하여야 한다.

다음은 서고가 프로그람완성에 어떻게 편입되는가를 구체적으로 보자. 특히 머리부파일을 참조하는 방법과 프로그람객체파일에서 서고정의를 련결하는 기구를 참조하는 방법을 서술한다.

문제

1. 2개의 파라메터를 가지는 scale이라는 함수원형을 지정하시오. 첫 파라메터는 int형이고 두번째는 double형이다. 함수의 복귀형은 int 형이다.
2. 파라메터를 가지지 않는 Blend라는 함수원형를 지정하시오. 돌림형은 double형이다.
3. 3개의 파라메터를 가진 Dither라는 함수를 선언하시오. 첫 파라메터는 double형이고 두번째 파라메터는 double형, 세번째는 float형이다. 되돌림값은 없다.

4. 프로그람작성자가 작성한 stats.h라는 파일을 프로그람에 포함시키는 파일포함지령을 주시오.
5. 프로그람에 iostream파일을 포함시키는 파일포함지령을 쓰시오.
6. 2개의 객체정의를 포함한 Object.h라고 하는 파일을 프로그람에 포함시키시오. 첫 객체는 0으로 초기화된 int형객체 count이다. 두번째 객체는 0으로 초기화된 double형객체 Average이다 . Object.h가 다른 모듈에 여러번 포함되여도 한번만 정의되도록 포함파일을 작성하시오.

5.3 쏘프트웨어서고의 리용

프로그람파일의 시작에 머리부파일을 포함하면 프로그람작성자는 프로그람파일전반을 통하여 머리부파일선언을 리용할수 있다. 프로그람파일전반을 통하여 리용될수 있는 선언을 전역선언(global declaration)이라고 한다. 모든 전역선언들의 모임을 전역이름공간(global namespace)이라고 한다. C++는 전역이름공간을 제공하는것외에 프로그람작성자에게 프로그람에서 정의된 이름공간에 선언을 할수 있는 능력을 준다. 부록 4에 이러한 이름공간을 창조하는 방법을 주었다.

이 장의 앞에서 주의를 준것처럼 선언들을 전역적으로 사용할수 있는 머리부파일은 보통 파일확장자 .h를 가진다. 그리고 선언을 비전역이름공간에 배치하는 머리부파일에서는 보통 파일이름에 확장자가 없다. iostream은 선언이 비전역이름공간에 배치된 머리부파일이다. 특히 이러한 파일에 있는 선언들은 이름공간 std에 배치된다. 이와 함께 rect.h는 EzWindows서고를 위한 머리부파일중에 하나인데 그 안에 있는 선언들은 특별한 이름공간에 할당되지 않는다. 따라서 그안에 있는 선언들은 전역이름공간이다.

전역이름공간에 선언된 객체와 자료형 혹은 함수는 특별한 문법이 없이도 참조될수 있다. 실례로 전역적으로 정의된 RectangleShape형을 리용하려면 그것을

```
RectangleShape R(W, 5, 4, blue);
```

와 같이 곧 리용할수 있다. 그러나 비전역이름공간에 있는 객체자료형, 함수를 참조하려면 일부 추가적인 문법이 필요하다. 이것을 처리하는 한가지 방법이 이름공간이름과 유효범위해결연산자 ::을 모든 참조에 붙이는것이다.

이것은 cout를 리용하는 다음의 명령문에서 볼수 있다.

```
std::cout << "Hello World " << endl;
```

두번째 방법은 요구되는 이름공간에 있는 모든 선언이 접근될수 있게 하는 using명령문을 리용하는 것이다.

```
using namespace std;
cout << "Hello world " << endl;
```

거의 모든 프로그람작성자들이 이 방법을 리용하는데 그것은 이름공간원소들을 매번 참조하는것이 합리적이지 못하기때문이다. 그러나 첫 방법은 객체형 혹은 문제의 함수원천을 더 엄밀하게 지정한다.

일반적으로 서고머리부파일에서는 함수정의를 하지 않는다. 대신 함수정의는 번역처리의 련결단계에서 프로그람에 조합되는 다른 파일에 위치한다. 프로그람에 서고객체모듈만을 련결한다면 콤파일처리는 빨라지게 된다. 그것은 서고는 이미 번역되여 있으므로 다시 번역할 필요가 없기때문이다. 번역처리는 1장에서 론의되였고 그림 5-2에 보여 주었다.

만일 서고가 표준서고라면 련결은 보통 콤파일러에 의해 자동적으로 진행된다. 서고가 표준서고가 아니라면 콤파일지령은 정의들이 발견될수 있는 곳에 지정되여야 한다. 서고의 련결은 콤파일러와 관련된 make나 project도구를 리용하여 자동화할수 있다. 부록 6에서 여러가지 콤파일러에 해당한 대상과제 창조와 조작방법을 취급한다.

다음절들에서 iostream서고와 여러가지 다른 서고들(iostream, iomanip, fstream과 assert)을 구체적으로 고찰하고 취급한다. 이러한 서고들에 정의된 함수, 클라스, 객체들은 이 장과 다음장의 실례들에서 리용된다. C++콤파일러는 또한 다른 표준서고들과 때로는 비표준서고들을 제공하기도 한다. 실례로 여러가지 종류의 목록들을 창조하고 례외처리, 접근시간과 날자정보의 얻기 등의 처리를 진행하는 표준서고들이 있다(9장에서 구체적으로 취급한다). 이 다른 서고들은 자주 리용되므로 부록외에 C++참고지도서나 콤파일러참고지도서를 리용해 주시오.

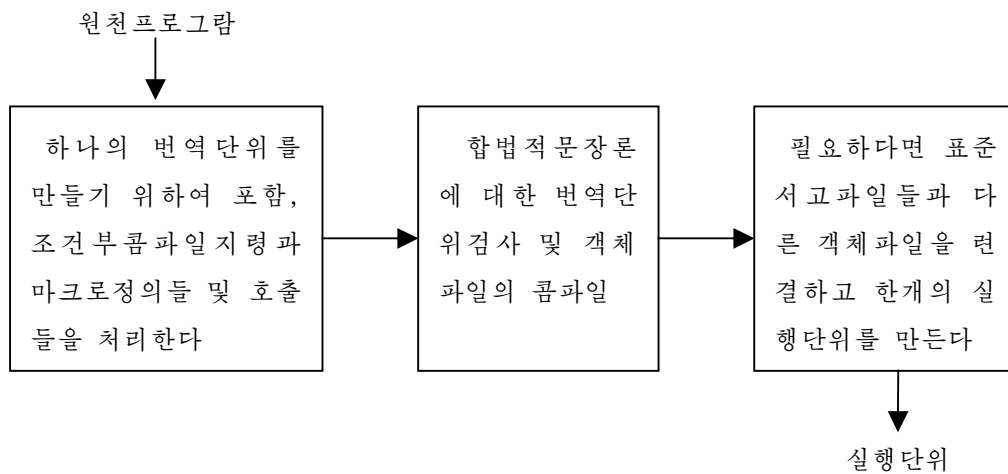



그림 5-2. 번역처리

5.4 iostream서고

많은 프로그람작성언어에서처럼 입력/출력(I/O)의 조작은 C++언어의 공식적인 부분이 아니다. 대신 I/O조작은 언어와 관련된 서고에 정의되여 있다. 입출력조작을 언어정의로부터 분리시키는것이 이상하게 보일수 있지만 그것은 쏘프트웨어개발자들이 과제를 위한 정확한 서고를 곧 선택할수 있게 한다.

C++ I/O서고는 콤퓨터체계를 구성하는 프로그람과 하드웨어장치사이의 대면부를 제공한다. 그 대면부는 2개의 추상화준위로 갈라 진다. 추상화의 낮은 준위에 파일이 있다. 여기서 파일이란 감시기, 모뎀이나 건반과 같은 개별적인 하드웨어장치를 표현하는 혹은 플로피디스크에 있는 파일이나 CD-ROM우에 있는 파일과 같은 하드웨어장치의 특별한 부분을 표현하는 론리적인 개념이다. 추상화의 높은 준위에 흐름(stream)이 있다. 흐름이란 실제적인 장치에 대한 하드웨어독립보기이다. 하드웨어에 혹은 하드웨어로부터 넘어 가고 넘어 오는 자료란 바로 바이트들의 렬이다.

흐름과 파일보기는 중요하다. 흐름보기(stream view)는 프로그람이 자료흐름에 관한 I/O의 일반적인 요구(그 요구들이 어떻게 수행되는가에는 무관계한)를 내보내도록 한다. 파일보기(file view)는 실제적인 장치의 중요한 물리적특성을 포착한다. 흐름보기의 I/O요구는 콤파일러에 의하여 요구를 해결하는 파일준위의 지정된 동작으로 자동적으로 번역된다.

iostream서고의 중요한 특성은 그의 확장성이다. 이 특성은 I/O서고의 하나인 stdio서고에는 없다. stdio서고는 C프로그람작성자가 I/O를 하는데 리용하는 서고이다. 표준형값과 문자렬에 대한 I/O요구를

처리하는 stdio함수들이 있지만 프로그람작성자가 정의한 자료형에 대하여 조작할수 있도록 이런 함수들을 확장하고 재정의하는 방법은 없다. 그리고 stdio에 기초한 입력처리에 지적자(11장에서 취급)들이 자주 요구되므로 많은 프로그람작성자들이 stdio를 리용할 때 오유를 범하게 된다. 이러한 리유로부터 iostream서고의 리용을 권고하며 그에 대해서만 론의한다.

iostream서고에서 기초 I/O기능을 계층적으로 제공하여도 여기서 중심은 흐름조작을 지원하는 계층의 웃준위이다.

5.4.1 표준흐름

머리부파일 iostream은 입력과 출력 혹은 입출력을 다 지원하는 흐름클라스들의 집합체이다. 이 집합체에는 프로그람실행을 시작할 때 자동창조되는 4개의 표준흐름객체가 있다. 지금까지의 모든 프로그람들에서는 이 표준흐름객체가운데서 cin 과 cout 2개만을 리용하였다. 흐름 cin의 실지형은 istream이고 cout는 ostream이다. 다른 2개의 표준흐름은 cerr와 clog이다. cout와 같이 2개의 표준흐름은 둘다 ostream이다.

5.4.2 표준오유흐름객체들

cerr와 clog흐름은 기정오유기록(error log)에 대한 출력을 현시한다. 여기서 기정오유기록은 보통 조종탁감시기이다. cerr와 clog가 흐름객체이므로 값, 성원함수, 성원연산자를 가진다. 특히 삽입연산자 <<는 그 성원들중의 하나이다.

프로그람의 비정상적인 상태를 검출하면 사용자에게 문제의 진상을 알리는 통보를 현시하는것이 좋다. 이러한 통보는 즉시에 볼수 있도록 보통 감시기에로 전달된다. cout흐름과 관련된 출력이 때때로 조작체계의 지령을 리용하는 파일에 다시 전달되기때문에 cout는 오유나 체계통보를 내보내는데는 적합치 않다. 즉 사용자는 출력파일을 찾을 때까지 기다려 이 통보들을 보려고 하지 않는다. 오유통보용으로 더 좋은 흐름이 바로 cerr이다. cout와 같이 그의 삽입요구는 기정으로 감시기에로 넘어 간다. 그러나 cout와는 달리 cerr는 보통 파일에 재전달되지 않는다.

cout와 cerr흐름은 또한 완충화에서도 차이난다. 완충화는 일반적으로 시간림계를 가지므로 오유통보용으로는 적합치 않다. 따라서 cerr흐름에 대한 삽입요구는 즉시에 현시장치에 보내여 진다.아래에 몇가지 오유통보의 실례를 주었다.

```
cerr << "체계가 2분동안 재시동중이다.\n";
cerr << "령나누기오유\n ";
cerr << "모뎀이 설치되지 않았다. 호출할수 없다.\n ";
```

효률상 리유로 완충된 오유체계상태통보를 가지는것이 필요하다면 완충된 오유흐름 clog가 사용되여야 한다. Cerr와 같이 clog는 보통 감시기에 전달된다. 아래에 몇가지 실례를 주었다.

```
clog << UserName << "has logged onto system\n ";
clog << "Backup was successfully performed\n ";
clog << "mail has arrived\n ";
```

5.4.3 iostream 조작자

iostream서고에는 또한 입력과 출력흐름을 조종하는 조작자들이 있다. 이 조작자들을 표 5-1에 주었

다. 조작자는 삽입출력연산자 <<과 추출입력연산자 >>의 오른쪽에 놓인다.

endl조작자는 이미 리용해 보았다. endl 조작자를 출력흐름에 삽입하였을 때 흐름에 행바꾸기문자를 추가하고 흐름이 진행되게 한다. ends조작자는 endl기호와 기능이 비슷한데 다른것은 흐름에 빈 문자인 '\0'을 삽입한다는것이다. 이 조작자는 문자렬처리에서 자주 리용된다.

dec, hex, oct기호는 관련된 출력흐름으로 현시되는 수값의 밑수를 지정한다. dec는 10진수(기정밑수)를 , hex는 16진수(밑수 16)를, oct는 8진수(밑수 8)를 지정한다. 이 세 조작자들은 지속적이다(즉 다른 수체계가 지정되기전까지 그 기능이 그대로 유지된다).

표 5-1. iostream조작자들

조작자	리 용
dec	소수점값을 현시한다
end1	행바꾸기를 진행한다
ends	빈 문자를 출력한다
flush	완충기에 축적한다
hex	16진수값을 현시한다
oct	8진수값을 현시한다

Flush조작자가 출력흐름에 지정될 때에는 그 이름그대로 즉 출력완충기를 씻어 낸다. 따라서 flush 연산은 행바꾸기문자를 삽입하지 않는다는 점을 제외하고는 endl과 같다. 다음의 실례는 이러한 조작자들을 리용한것이다.

```
int i = 10;
int j = 20;
int k = 30;
cout << i << " " << j <<" " << k << endl;
cout << I << " " << j << " " << k << "\n " << flush;
cout << oct << i << " " << j << " " <<k << endl;
cout << hex<< i <<" " << j <<" " << k << endl;
```

처음 2개의 출력명령은 둘다 i, j, k를 10진수형식(기정)으로 출력한다는 점에서는 같다. 단지 행바꾸기문자가 현시되는 방법이 차이난다. 첫번째 출력명령문에서 endl조작자가 행바꾸기문자를 쓰고 출력완충기를 플라쉬하지만 두번째 명령문에서는 행바꾸기문자를 반드시 삽입하고 플라쉬조작자를 불러 낸다.

세번째 삽입명령문에서는 oct조작자를 리용하였는데 이것은 그다음에 오는 i, j, k를 8진수형식으로 현시하게 한다. 마지막삽입명령문에서 사용한 hex조작자는 i, j, k를 16진수형식으로 현시하게 한다. 우의 코드토막의 실행결과는 아래와 같다.

```
20 30
10 20 30
12 24 36
a 14 1e
```

5.5 iomanip서고

iomanip서고에는 출력과 입력동작을 수정하는 I/O흐름조작자들의 집합이 정의되여 있다. 이 조작자들은 흐름객체들과 련결된 다른 함수들이 이 과제들을 수행할수 있기때문에 반드시 필요하지는 않다. 그러나 이 다른 함수를 호출하는것은 iomanip조작자를 사용할 때보다 일반적으로 불편하다.

iomanip조작자는 표준머리부파일인 iomanip파일에 정의된다. 실천에서는 C++프로그람들이

iostream머리부파일과 함께 iomanip파일을 포함하도록 하는것이 일반적이다.

```
#include <iostream>
#include <iomanip>
```

iomanip이 제공하는 조작자들의 선택된 목록을 표 5-2에 주었다. setw()조작자를 제외하고 모든 조작자들은 지속적이다. 한번 정의되면 다른 조작자가 정의될 때까지 그 기능이 그대로 유지된다. 일부 이러한 조작자들을 실현하지 못한 콤파일러들도 있다.

표 5-2. iomanip 조작자들

출력조작자	기 능
setw(int w)	마당폭을 w로 설정
setfll(int c)	채우기문자를 c로 설정
left	현시후 채우기문자들이 넣어 진다
right	현시전에 채우기문자들이 넣어 진다
setbase(int b)	수표기체계를 b로 설정
fixed	류점수를 소수점방식으로 출력
scientitic	류점수를 과학표기로 출력
showpoint	류점수가 언제나 소수점을 가지고 현시된다
noshowpoint	소수점값이 0이 아니라면 류점수값을 소수점값으로 바꾼다
setprecision(int d)	정확도자리수를 d로 설정
skipws	추출할 때 공백문자가 무시
noskipws	공백문자가 추출가능하다
showpos	어깨수가 +기호를 가진다
noshowpos	어깨수가 +기호를 가지지 않는다
showbase	8진수현시는 0을, 16진수는 0x를 앞에 붙힌다
noshowbase	기준수지적자가 현시되지 않는다
boolalpha	론리값들을 기호적으로 즉 true와 false로 현시한다
noboolalpha	론리값들을 0과 1로 현시한다
resetiosflags(long f)	f로 지적된 기발들을 0으로 설정한다
setiosflags(long f)	f로 지적된 기발들을 1로 설정한다

setw()조작자는 출력폭을 설정한다. 앞서 이야기하였지만 많은 프로그람작성자들은 setw()조작자가 지속적으로 되지 않는다는것을 아쉬워 한다. 조작자에 지정한 폭은 오직 그 다음번 삽입에만 영향을 준다. 다음번 삽입이후에 기정동작이 다시 리용된다.

setw()에 요구한 폭이 다음값을 현시하는데 지내 작다면 다음의 출력의 폭은 정확히 현시할수 있는 최소한계까지만 증가시킨다. 어떤 값을 현시하는데 필요한 최소폭을 값의 형에 따라 결정한다. 특히 문자렬에 대하여 setw()가 지정한 폭이 두 문자렬길이보다 작다면 문자렬길이가 그대로 폭으로 리용된다. 옹근수에 대해서는 필요하다면 모든 수들에 부호까지 붙여서 현시한다. 류점수값인 경우는 과학적인 표기법이 리용되는가 아니면 10진표기법이 리용되는가에 따라 필요한 폭이 변한다. 두 경우에 다 최소폭은

옹근수부와 소수부를 다 포함할수 있으므로 과학적인 표기인 경우에는 지수를 리용한다. setw()가 지정하는 폭이 필요이상으로 크다면 오른쪽을 기준으로 하여 수를 현시하되 나머지폭에 해당하는 자리는 채우기문자로 가득 채운다. 채우기문자는 기정값으로 공백이다.

다음의 "Hello World"라는 문자렬의 삽입을 고찰하시오.

```
cout << setw(1) << "Hello World" << endl
  << setw(15) << "Hello World" << endl
  << "Hello World" <<endl;
```

삽입은 출력으로 나온다.

```
Hello World
      Hello World
Hello World
```

두번째 행의 출력결과는 첫번째 행과 다른데 그것은 문자렬현시에 앞서 진행한 setw()의 호출을 서로 다르게 하였기때문이다. 첫번째로 호출한 setw(1)은 문자렬현시에 필요한 폭보다 작게 설정하였으므로 현시폭은 문자렬길이만큼 증가된다. 두번째로 호출한 setw(15)는 문자렬의 길이보다도 폭을 설정하므로 오른쪽을 기준으로 현시한다(즉 문자렬의 앞에 채우기문자 4개를 현시한다). 마지막행의 결과를 보면 행의 첫 위치로부터 문자렬을 현시한다. 그것은 setw()조작자의 기능이 지속성을 가지지 않기때문이다.

Setfill()조작자는 삽입폭요구가 필요이상으로 커질 때 현시되는 채우기문자를 설정한다. 다음의 명령문은 이 조작자의 리용실례이다.

```
cout << setfill('#') << setw(15) << "Hello" << endl;
```

이 삽입명령문에서 문자렬 "Hello"가 '#'문자가 현시될 때 오른쪽으로 조절된다.

```
##########Hello
```

left 와 right조작자는 삽입이 오른쪽맞추기방식으로 현시되는가 아니면 왼쪽맞추기방식으로 출력되는가를 조종한다. 다음의 실례는 6문자폭으로 현시되는 3명의 이름을 현시하는 명령문들이다.

```
cout << ":" << left << setw(6) << "JJ" << ":" << endl;
cout << ":" << right << setw(6) << "Jenna" << ":" << endl;
cout << ":" << left << setw(6) << "Hannah" << ":" << endl;
```

출력된 결과는

```
:JJ :
:Jenna :
:Hannah :
```

이다. 첫번째 결과행을 보면 JJ라는 문자렬이 현시되고 다른 4개의 공백이 현시된다. 이것은 left조작자가 있기때문에 이름뒤에 공백이 생긴다. 문자렬 "Jenna"는 길이가 5문자이므로 공백이 하나이다. 그리고 문자렬 "Hannah "는 6문자로서 출력폭과 같기때문에 아무런 조종도 수행하지 않는다. left나 Right조작자는 표형식의 출력을 진행할 때 주로 리용한다.

setbase()조작자는 출력흐름에 쓰이며 하나의 파라메터를 요구한다. 그 파라메터는 수값자료의 현시에 리용할 밑수(10진, 8진, 16진)를 지정한다. 이 조작자는 iostream조작자들인 dec, oct 그리고 hex

도 사용할수 있기때문에 일반적으로 리용되지 않는다. 보통 setbase()는 어떤 값을 삽입하기 위하여 요구되는 밑수가 이전 계산 혹은 입력요구의 결과일 때에만 리용한다. 아래의 실례는 의도한것과 다른 결과를 준다.

```
int number;
int base;
cout << "Provide a number and a base;";
cin >> number >> base;
cout << number << "in decimal is" << setbase(base) << number
  << "in base" << base << endl;
```

number와 base의 입력값이 9와 8이라면 결과는

```
9 in decimal is 11 in base 10
```

이다. 이 결과는 기대하던 결과가 아니다. 마지막에 있는 10은 8을 8진수로 현시한것이다. 이러한 현상은 setbase( )호출의 효과가 그대로 유지되기때문에 생긴다. 즉 setbase()호출후에 생긴 number와 base에 대한 cout삽입은 8진수로 현시된다. 의도하던 결과를 얻으려면 다음과 같이 수정해야 한다.

```
cout << number <<"in decimal is"
  << setbase(base) << number << "inbase" << dec
  << base << endl;
```

출력결과는 요구대로 된다. 즉

```
9 in decimal is 11 in base 8
```

만일 number와 base 에 19와 16을 입력한다면 결과는 다음과 같다.

```
19 in decimal is 13 base 16
```

fixed와 scientific조작자는 류점수값이 소수점표기법으로 되여야 하는가 과학적표기법으로 현시되여야 하는가를 지정한다. 기정적으로 류점수값은 자동방식으로 현시된다. 자동방식인 경우 큰 값, 작은 값, 령에 근사한 값은 과학적인 표기법으로 현시되며 한편 다른 값들이 소수점표기법으로 현시된다. 실례로 다음의 코드토막

```
cout << 1000000000000.0 << endl;
cout << 0.0000000000001 << endl;
cout << 921.28 << endl;
cout << -100000000000.0 << endl;
```

은 자동방식에서 다음과 같은 출력결과가 현시된다.

```
1e+012
1e-013
921.28
-1e+012
```

그러나 고점수방식을 설정하면 다음의 코드토막

```
cout << fixed << 1000000000000.0 << endl;
cout << 0.0000000000001 << endl;
cout << 921.28 << endl;
cout << -1000000000000.0 << endl;
```

의 결과는

```
1000000000000.000000
0.000000
921.28000
-10000000000.000000
```

이다. 고찰해 보면 고정방식은 소수점아래 6자리까지 정확도를 보장한다. 대신 파학적방식에서 다음의 코드토막

```
cout << scientific << 1000000000000.0 << endl;
cout << 0.0000000000001 << endl;
cout << 921.28 << endl;
cout << -100000000000.0 << endl;
```

의 결과는

```
1.000000e+012
1.000000e-013
9.212800e+002
-1.000000e-012
```

이다. 고찰해 보면 파학적인 방식 역시 6자리까지 정확도를 보장한다. setprecision()조작자는 류점수를 현시할 때 정확도자리수를 설정한다. 자동방식에서는 정확도값이 현시될 자리수와 갈다. 고점수와 파학적인 방식에서는 정확도값이 소수점아래 자리수이다. 자동방식인 경우 주의해야 할것은 출력할 때 마지막에 있는 령(0)들은 현시하지 않는다는것이다. 다음의 실례가 이것을 보여 준다.

```
cout << setprecision(6)
  << 12.01234 << endl
  << 12.0123 << endl
  << 12.012 << endl
  << 12.01 << endl
  << 12.0 << endl
```

결과는

```
12.01234
12.0123
12.012
12.01
12.0
```

이다. 처음 2개 행은 setprecision(6)에 의하여 6자리정확도를 보장하므로 출력결과가 같다. 이것은 류점수값의 현시가 항상 대략적으로 취급된다는것을 보여 준다. 사실 값을 유한정확도로 현시하여야 한다고 가정하면 사용자는 소수점아래 몇개의 자리수가 현시되여도 일이 없겠는지 항상 주의해야 한다. 다음 세개 행은 기정으로 0이 아닌 마지막수자만이 현시된다는것을 보여 준다. 또한 마지막행은 모든 소수점자리가 0이면 소수점도 기정으로 현시되지 않는다는것을 보여 준다. 그러나 showpoint조작자를 리용하여 소수점을 현시할수 있다. 이 조작자의 의미는 noshowpoint조작자로써 해제한다. 다음의 실례

```
cout << setprecision(0) << 12.01234 << endl;
```

을 고찰해 보면 삽입명령문의 결과는 12.01234이다.

setprecision( ) 파라메터가 0이면 자동적인 정확도를 보장하기때문에 이런 결과가 생긴다. 이번에는 류점수 10.12345를 현시하는 삽입명령문을 고찰하자.

```
cout << setprecision(5)
  << setw(5) << 10.12345 << endl
  << setprecision(4)
  << setw(9) << 10.12345 << endl
  << setw(5) << 10.12345 << endl;
```

첫번째로 삽입되는 10.12345는 요구되는 정확도가 5이고 전체 현시폭은 5이다. 5자리정확도요구가 선행하였기때문에 5자리수와 소수점을 포함한 현시를 할수 있도록 전체 현시폭이 6으로 증가된다.

```
10.123
```

다음의 삽입은 setprecision(4)이므로

```
10.12
```

이다. 이때 10.12345의 삽입은 요구되는 전체 현시가 지금은 9이기때문에 첫번째 결과와는 차이난다. 폭이 필요이상이기때문에(4자리정확도가 작용한다고 하면) 오른쪽을 기준으로 값이 현시되고 나머지는 채우기문자로 채워 진다. setw()가 계속 유지되지 못하므로 다음의 결과는 행의 첫 위치에서 시작하며 setprecision()은 유지되므로 4가지 정확도로 현시된다. 즉

```
10.12
```

이다. 123456789.123456789와 같이 큰 류점수값에 대해서는 과학적표기법이 기정값에 의하여 리용된다. 다음의 삽입명령문을 보면

```
cout << setw(6) << 123456789.123456789 << endl
  << setprecision(3)
  << setw(15) << 123456789.123456789 << endl;
```

의 결과는

```
1.23457e+08
        1.23e08;
```

이다. 두 행의 차이는 setw()와 setprecision()때문에 생긴다. 첫번째 삽입은 현시될 값이 둥그리기되였다(둥그리기는 표준방법을 쓴다). 지속적조작자인 boolalpha와 noboolalpha는 bool객체가 어떻게 현

시되는가를 지정한다(기정적으로는 bool객체가 2진표기로 현시된다). 다음의 실례

```
cout << true << endl;
cout << false << endl;
```

의 결과는

```
1
0
```

이다. 문자렬표기를 리용하자면 boolalpha조작자를 리용한다. 실례로

```
cout << boolalpha << true << endl;
cour << false << endl;
```

라고 코드를 작성하면 결과는

```
true
false
```

이다. 2진표기로 다시 돌아 가려면 noboolalpha조작자를 리용한다. showbase와 noshowbase지속조작자는 8진수와 16진수를 식별하는 앞붙이가 붙어 현시되는가 아닌가를 지정한다(기정값은 식별앞붙이가 없다). showbase를 리용하면 8진수는 0이,16진수인 경우는 0x가 붙어 현시된다. 앞붙이의 사용은 noshowbase조작자로 해제할수 있다.

showpos 와 noshowpos는 10진수 아닌 현시를 하는것을 정수로 한다. 이 동작은 noshowpos 조작자를 리용하여 해제한다. 수식별조작자는 다음의 코드토막들에서 볼수 있다.

```
cout << oct << showbase << 8 <<" " << hex << 8 << endl;
cout << dec << showpos << 8 <<" " << noshowpos << 8 << endl;
```

결과는 아래와 같다.

```
010   0x8
+8   8
```

skipws와 noskipws 조작자는 입력흐름조작자이다. 이 종기들은 공백이 무시되는가 아닌가를 조종한다. 앞서 고찰한것처럼 C++의 기정적인 방식은 공백을 무시하는것이다. 이 조작자들은 다음의 코드토막에서 볼수 있다.

```
char s1,t1,s2,t2,s3,t3;
cout << "Enter text:";
cin >> s1 >> t1;
cout << "[" << s1 << "]" << endl;
cout << "[" << t1 << "]" << endl;
cin >> noskipws >> s2 >> t2;
cout << "[" << s2 << "]" << endl;
cout << "[" << t2 << "]" << endl;
cin >> skipws >> s3 >> t3;
```

```
    cout << "[" << s3 << "]" << endl;
    cout << "[" << t3 << "]" << endl;
```

abcde를 입력하면 다음의 결과가 생긴다.

```
Enter text: a b c d e
[a]
[b]
[c]
[d]
[e]
```

공백입력이 초기에 무시되였기때문에 이런 결과가 생긴다. 따라서 s1과 t2에는 a와 b사이의 공백이 무시되여 그 값은 'a'와 'b'이다. noskipws조작자에 의하여 c의 앞에 있는 공백이 입력된다. 따라서 객체 s2와 t2에는 ' '과 'c'가 들어 간다. 그다음 skipws조작자가 사용되여 공백은 다시 무시되였다. 따라서 객체 s3과 t3은 'd'와 'e'으로 된다.

5.6 fstream서고

4장에서 표준입력흐름 cin으로 수를 추출하여 그의 평균값을 표준출력흐름 cout로 출력하는 프로그람을 작성하였다. 그 프로그람은 아래와 같다.

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    cout << "Please provide list of numbers" << endl;
    int ValuesProcessed = 0;
    float ValueSum = 0;
    float Value;
    while (cin >> Value){
        ValueSum += Value;
        ++ValuesProcessed;
    }
    if (ValuesProcessed > 0) {
        float Average = ValueSum /ValuesProcessed;
        cout << "Average:" << Average << endl;
    }
    else
        cout << "No list to average" << endl;
    return 0;
}
```

10개정도의 값에 대하여 프로그람을 동작시키면 적합하다. 그런데 훨씬 많은 수를 처리하였더라면 아마 파일로부터 그 값을 얻는 프로그람이 비슷해 질것이다. 파일에서 값들을 읽어 들여 그의 정확성을 검사할수 있고 필요하다면 틀린 값을 고칠수도 있다. 이와 함께 프로그람이 많은 량의 출력을 생성한다면 아마 정보를 다시 볼수 있게 파일에 출력하는 편이 나을것이다. 표준서고인 fstream을 리용하여 이러한 파일입력과 출력을 진행할수 있다. 서고의 대면부는 표준머리부파일 fstream에서 지정한다.

fstream서고에는 흐름클라스형들인 ifstream, iofstream 그리고 fstream이 정의되여 있다. 클라스 ifstream은 ifstream에서 파생된다. 클라스 ifstream은 파일로부터 값을 읽어 들이기 위한 입력흐름들을 창조한다. Ostream클라스는 iostream에서 파생된다. ostream클라스는 값들을 파일에 삽입하기 위한 추출흐름을 창조한다. 클라스 fstream은 iostream클라스에서 파생되였는데 입력과 출력능력을 다 가지고 있는 파일지향흐름을 정의하기 위한것이다.

mydata.nbr라는 파일에서 값을 입력하는 fin이라는 입력흐름이 필 요하다면 요구되는 흐름은 초기화에서 "mydata.nbr"라고 하는 문자렬을 리용하는 ifstream객체 fin을 정의하여 창조될수 있다.

```
ifstream fin("mydata.nbr");
```

정의에서 보는것처럼 처리하려는 파일의 이름을 문자렬로 넘긴다.이 정의는 흐름 fin 과 mydata.nbr 파일사이의 일치성을 설정한다.

ifstream클라스는 istream에서 파생되였기때문에 istream추출연산자 >>가 ifstream객체에 대하여 정의된다. 그러나 추출연산자가istream 객체에 적응될 때 표준입력흐름 cin이 아니라 결합된 파일에서 자료가 나온다.

iostream입력조작자들과 성원함수들은 물론 iomanip입력조작자를 ifstream 객체에 리용할수 있다. 이와 비슷하게 iostream출력조작자와 성원함수,iomainp출력조작자들은 ofstream객체에서 리용할수 있다. 실례로 다음의 명령문은 파일 mydata.nbr에서 값을 입력한다.

```
fin >> Value;
```

ofstream객체를 정의하면 파일에 삽입을 포착하기 위하여 그것을 리용할수 있다. 아래의 실례에서는 average.nbr파일과 련결된 ofstream 객체 fout를 련결하고 그 파일에 입력의 평균값을 삽입한다. 기정적으로 ofstream객체에 련결된 파일은 객체를 초기화할 때 내용이 다 지워 진다. 이런 기정동작을 절단방식이라고 한다.

```
ofstream font ("average.nbr");
fout << Average << endl;
```

현재의 내용을 잃어 버리지 않으려면 흐름정의에서 나머지파라메터를 지정해야 한다. 특히 나머지파라메터는 추가방식이 작용한다는것을 나타내야 한다. 추가방식으로 삽입은 즉시 현재 존재하는 파일내용 바로 뒤에 추가된다. 이 파라메터를 표현하는 방법은 (ios-base::out|ios-base:: app)식을 리용하는것이다.

식은 파일이 쓰기형식으로 열리고 삽입은 파일의 끝에 추가된다는것을 가리키는 2개의 선택항목을 지정한다. 이와 같은 방법으로 다음의 정의에서 출력흐름 myout는 파일 dataset.txt와 련결된다.

```
ostream myout("dataset.txt",(ios-base::out | ios-base::app));
```

fin과 fout흐름정의를 리용하여 우에서 실례 든 평균계산프로그람 mydata.nbr파일을 값을 추출하고 평균값을 계산하여 average.nbr파일에 쓰기하는 프로그람으로 수정할수 있다. 그 프로그람을 프로

그람 5-2에 주었다.

```
// 프로그람 5-2: mydata,nbr의 평균계산
lude <fstream>
lude <string>
namespace  std ;
main() {
    ifstream  fin ( "mydata.nbr") ;
    int ValuesProcessed = 0 ;
    float ValueSum = 0 ;
    float Value ;
    while (fin >> Value) {
        ValueSum += Value ;
        ++ ValuesProcessed ;
    }
    if (ValuesProcessed > 0) {
        ofstream fout ("average.nbr") ;
        float Average = ValueSum / ValuesProcessed ;
        fout << "Average :" << Average << endl ;
    }
    else {
        cerr << "평균할수 없음" << endl ;
    }
    return 0 ;
}
```

프로그람 5-2. 파일자료의 평균값계산.

프로그람 5-2에서 while순환은 fin흐름에서 더이상 값을 추출하지 못할 때까지 반복된다. 순환식 fin>>Value는 cin>>Value와 같은데 추출이 성과적으로 되였을 때 true로 평가된다. 추출이 성과적으로 되지 못하면(즉 추출할 자료가 더이상 없다.) false로 평가된다. 이때 이러한 속성을 가진 흐름이 파일끝(eof)에 도달하였다는것을 의미한다. 때때로 입력 및 출력파일이 어떤 리유로 하여 잘못 지원되거나 사용할수 없는 경우가 있다. 이런 경우를 검사하는 방법이 있다. 흐름이 사용될수 있다면 령 아닌 값(즉 true)을 흐름이 가지며 그렇지 않으면 령값(false)을 가진다. 다음의 실례는 추출하기전에 입력파일을 사용할수 있는가 시험하기 위하여 if명령문을 리용한다.

```
ifstream fin ("mydata.nbr") ;
int Number ;
if (fin) {
    fin >> Number ;
    cout << Number << endl ;
```

```
else {
    cout << "mydata.nbr는 사용할수 없다." << endl ;
    Number = 1 ;
}
```

파일처리에서 보다 좋은 유연성을 보장하려면 파일이름을 실행시 지적하고 그 파일을 처리해야 한다. 파일이름의 추출은 string형을 리용하여 쉽게 할수 있다. 그러나 fstream서고는 string형을 인식하지 못하므로 다음의 실례코드는 콤파일되지 않는다.

```
string s ;
cout << "파일이름 입력:";
cin >> s;
ifstream fin(s);
```

그러나 string객체는 문자렬현시를 돌려 주는 성원함수 c_str( )를 가지고 있는데 이 함수는 약속된 문자렬이 예견되는 곳마다 리용할수 있다. 특히 c_str()성원함수가 돌려 주는 값은 ifstream이나 ofstream객체초기화에서 유효하다. 따라서 입력흐름 fin을 초기화하는 정확한 방법은 다음과 같다.

```
string File Name;
cout<< "Enter name of file to process:";
cin>> File Name ;
ifstream fin(File Name, c_str());
```

fin을 초기화한다면 추출값이 유효한 흐름인가 검사하는것이 좋을것이다. 우에서 언급한것처럼 그러한 시험은 ifstream객체의 값을 조사하여 할수 있다. 다음의 코드토막은 그러한 시험을 수행한다 .

```
if (! fin ) {
    cerr << "Cannot Open" << File Name << " for averaging." << endl ;
    exit (1) ;
}
// fin에 의해서 현시된 파일흐름을 처리할 준비를 한다.
```

초기화가 잘 되지 않으면 오유통보가 생기고 프로그람이 완료된다.

exit()함수는 c에서 쓰는 stdlib서고에 정의되여 있고 그것은 다양한 편의함수의 집합이다. 이 서고에 대한 머리부파일은 cstdlib이다. 함수 exit()는 호출되였을 때 프로그람을 즉시 끝낸다. exit()의 파라메터는 프로그람의 되돌림값으로 리용된다. 따라서 코드토막의 끝에 있는 설명에 이르자면 식 (!fin)이 false이고 입력흐름 fin이 처리할수 있는 파일을 표현하여야 한다. 프로그람 5-2에 이 코드토막을 좀더 추가하여 일반목적평균계산프로그람을 창조할수 있다. 그 프로그람을 프로그람 5-3에 주었다.

Ifstream과 ofstream형객체들은 여러개의 성원함수들을 가지고 있다. 관심사로 되는 2개의 성원함수들은 close()와 open()이다. close()성원함수는 흐름에 적용될 때 련결된 파일의 처리가 완수되였다는것을 가리킨다. 실례로 다음의 명령문은 현재 fin과 fout에 련결된 파일에 더이상 삽입이나 추출이 없다는것을 지적한다.

```
fin close ( );
fout close ( );
```

close()는 보통 그 흐름에 다시 새로운 파일을 련결할 때에만 호출된다.

```
// 프로그람 5-3: 파일을 입력하고 평균값을 계산한다.
#include <fsteam>
#include <iostream>
#include <cstdlib>
#include <string>
using spacename std;
int main() {
    cout << "File of values to be averaged: ";
    string Filename;
    cin >> Filename;
    ifstream fin(FileName.c_str());
    if (! fin) {
        cerr << "Cannot open" << Filename;
        << " For  averaging." <end1;
        exit(1);
    }
    int ValuesProcessed=0;
    float ValueSum=0;
    float Value;
    while (fin>>Value){
        Valuesum+=Value;
        ++ValuesProcessed;
    }
    if(ValuesProcessed > 0) {
        float Average = ValueSum / ValuesProcessed;
        cout << "Average of values from"0<< FileName
          << " is "<< Average << end1;
    }
    else {
        cerr<< "No values to average in" << Filename << endl;
        exit(1);
    }
    return 0;
}
```

프로그람 5-3. 일반목적평균값계산기

성원함수 open()에는 개념적으로 두가지 변종이 있다. 두가지가 다 흐름에 련결된 파일이름을 표현

하는 문자렬을 파라메터로 요구한다. 2개의 변종가운데서 하나는 두번째 파라메터가 파일열기방식을 지정하도록 한다.

다음의 코드토막은 2개의 서로 다른 파일을 처리하는 ifstream객체 sin을 리용한다. 첫번째 파일 in1.txt는 sin을 정의할 때 sin과 련결된다. 두번째 파일 in2.txt는 open()호출의 결과로서 sin과 결합된다. 2개의 파일처리는 아주 간단하다. 첫번째 파일은 sout로 삽입을 하여 out1.txt에 복사한다. Sout 흐름은 out1.txt와 절단방식으로 리용하는 정의에 의하여 련결된다.

주의

2중역사선

다음의 객체정의를 살펴 보자.

```
ifstream  InStream ("num\n br.txt") ;
```

Windows에 익숙된 사람들은 객체 InStream과 련결되는 파일이 nbr.txt파일이고 이것은 num등록부에 있을것이라고 생각한다. 그러나 그 생각은 잘못된것이다. 역사선문자렬이 전처리에 의하여 해석될 때 파일이름에서 \n은 행바꾸기문자로 해석된다. 따라서 문제의 파일은 새행을 4번째 문자를 포함한다고 가정한다. 행바꾸기문자는 MS-DOS파일이름으로는 잘못된 문자이므로 InStream은 잘못 초기화된다. 초기화를 정확히 하려면 다음의 명령문과 같이 \\를 리용하여야 한다.

```
ifstream InStream ("num\\n br.txt") ;
```

두번째 파일은 추가방식을 지정하는 open()함수에 의하여 sout로 련결된 파일 out2.txt에 복사를 진행한다.

```
ifstream sin ("in1.txt");  //in1.txt에서 추출
ofstream sout ("out1.txt");  //out1.txt에 삽입
string s;
while ( sin >> s ) {
    sout << s << endl ;
}
sin close( );
sout close( );
sin.open  ("in2.txt") ;
sout.open ("out2.txt", (ios_base :: out | ios_base::app ));
while ( sin >> s ) {
    sout << s << endl;
}
sin.close ( );
sout.close ( );
```

sin이 in2.txt와 련결되기전에 in1.txt와의 련결이 close()호출에 의하여 완료된다. 이와 같이 out2.txt와 sout가 련결되기전에 out1.txt와의 련결은 close를 호출하여 완료된다. 분리는 int1.txt와 out2.txt의 고유한 처리에 필요하다.

5.7 란 수

란수렬은 많은 콤퓨터응용프로그람에서 중요한 역할을 한다. 실례로 사람들이 유희를 놀 때마다 같은 상태에서 시작되지 않도록 유희를 작성하는데 이 란수렬이 리용된다. 란수렬은 또한 새로운 고속도로와 같은 복잡한 체계를 설계하는데 리용된다. 고속도로를 구축하기전에 보통 콤퓨터로 모형화하고 모의한다. 란수렬로써 도로상에 있는 자동차의 움직임과 충돌을 모의하여 골목길이나 교차도로에서 충돌을 평가할수 있다.

특별히 리용되는 란수렬이 바로 균등한 수렬이다. 균등한 수렬은 어떤 지정된 수모임이 있어서 그로부터 란수를 취하거나 꺼내여 이루어 진다. 란수렬의 매개 위치에서는 그 모임중 임의의 수가 균일하게 나타날것이다.

가령 수모임 {1, 2, 3, 4, 5, 6 }에서 균등하게 분포된 란수렬을 창조한다고 가정하면 6면체주사위를 리용하는것이 한가지 방법으로 되여 있다. 주사위를 던질 때 6개면중 하나가 나타난다. 이때 모든 면들은 똑 같은 확률로 나타난다. 이렇게 얻어 진 수렬은 수모임 {1, 2, 3, 4, 5 ,6}에서 얻은 균등한 란수렬로 된다.

란수렬은 우연적이므로 정확한 란수렬을 반복적으로 계산해 내는 알고리듬은 있을수 없다. 알고리듬을 구성하는 명령들은 결정론적규칙이다. 즉 그 규칙은 다음변수를 알수 있게 한다. 그러나 일부 함수들을 리용하면 우연적인것으로 보이는 수렬을 만들수 있다. 이러한 수렬을 적당히 모조란수렬이라고 한다.

C++표준서고인 stdlib서고는 모조란수렬을 만드는 2개의 함수를 제공한다. 하나는 rand(), 다른 하나는 srand()인데 둘 다 stdlib.h에 선언되여 있다. rand()함수는 파라메터가 없다. 이 함수를 호출할 때마다 함수는 간격 0부터 RAND_MAX까지 동일한 모조란수렬를 돌려 준다. 여기서 RAND_MAX는 stdlib.h에 정의되여 있으며 실행에는 무관계한 전처리마크로상수이다. Rand()의 대다수실현에서 현행 모조란수의 발생은 이미 발생된 모조란수의 함수이다. 어떤 프로그람에 의한 첫 모조란수의 발생은 seed라는 초기값과 비슷한 함수에 기초한다. 이 초기값이 모조란수발생기로 제공된다. 프로그람 5-4는 함수 rand()의 반복호출에 의한 5개의 모조란수를 현시한다.

```
// 프로그람 5-4: 란수의 현시
#include <iostream>
#include <string>
#include <stdlib.h>
using namespace std;
int main() {
    for ( int i = 1; i <= 5; ++i )
    cout << rand() << endl;
    return 0;
}
```

프로그람 5-4. srand()을 리용하지 않고 5개의 란수를 발생하고 현시하는 프로그람

프로그람은 RAND_MAX가 32,767과 같은 체계상에서 실행되였다. 첫번째로 실행결과는 다음과 같다.

346

130
10982
1090
11656

프로그람을 두번째로 실행하였을 때의 결과는 다음과 같다.

346
130
10982
1090
11656

두 실행결과가 같은것은 rand()설계에 잘못이 있는것이 아니다. 사실 반복은 함수설계의 한 부분이다. 기정적으로 rand()는 늘 같은 모조란수렬을 만든다. 따라서 프로그람을 시험하거나 검사할 때에 같은 명령문실행순서를 다시 만들수 있다. 모조란수의 각이한 렬을 만들기 위하여 srand()함수를 리용한다. srand()함수는 unsigned int형파라메터를 가진다(unsigned는 부호비트를 가지지 않는 수를 의미한다. 즉 모든 값들이 정수이다). 파라메터를 리용하여 첫번째 모조란수를 발생하기 위한 초기값(seed)을 설정한다. 한번 초기값이 설정되면 rand()는 각이한 모조란수렬을 만든다. 프로그람 5-4를 수정한 프로그람을 프로그람 5-5에 주었다. 새 프로그람에서 사용자는 초기값을 주고 그것을 srand()에 넘겨야 한다.

```
// 프로그람 5-5:  5개의 란수를 현시
#include <iostream>
#include <string>
#include <stdlib.h>
using namespace std;
int main() {
    cout << "Random number seed (number): ";
    unsigned int seed;
    cin >> seed;
    srand(seed);
    for (int i = 1; i <= 5; ++i )
       cout << rand() << endl;
    return 0;
}
```

프로그람 5-5. srand()에 대한 입력재촉값을 리용한 5개의 란수현시

아래에 프로그람 5-5의 입출력동작실례를 준다.

Random number seed (number) = 12
4155
30526

32049

23353

29576

아래에 또 다른 실례를 주었다.

Random number seed (number) = 13

4501

30310

10132

13461

7374

프로그람 5-5에 초기값발생이 자동적으로 되게 하는 부분을 새롭게 수정하여 프로그람 5-6에 주었다. 프로그람 5-6은 현재시간을 초기값으로 리용한다. 현재시간을 초기값으로 리용하면 프로그람은 매번 실행할 때마다 초기값이 차이나며 란수렬도 각이하게 얻어 진다.

현재 시간을 time()함수로 결정하는데 이 함수는 표준 time 서고에 정의되여 있다. time()함수는 time_t형의 값을 되돌리는데 이 자료형은 또한 time서고에 정의되여 있는 옹근수형이다. time()함수에 넘겨 진 값이 령이면 time()함수는 그 되돌이값으로 현재의 시간을 지원한다.

```
// 프로그람 5-6: 모조란수현시
#include <iostream>
#include <string>
#include <stdlib.h>
#include <time.h>
using namespace std;
int main() {
    srand((unsigned int) time(0));
    for(i = 1; i <= 5; ++i )
        cout << rand() <<end1;
    return 0;
}
```

프로그람 5-6. 각이한 모조란수렬의 자동현시

프로그람 5-6의 입출력동작실례를 아래에 주었다.

1372

20765

3967

26247

6928

또 다른 실행결과는 다음과 같다.

```
7084
8706
20276
6944
8322
```

프로그람 5-6에서 다음의 명령문은 C++ 형변환식의 실례이다.

```
srand ((unsigned int) time(0));
```

형변환식은 콤파일러에 그의 연산수값을 정확한 형으로 변환할것을 지시한다. 이 명령문에서 식은 콤파일러에 time()에로의 호출결과를 time_t형인 time()에로의 호출결과의 unsigned int로 변환하도록 지시한다. 형변환식은 여러가지로 리용된다. 방금 그 하나를 보았다. 이 실례에서는 형변환식이 실지 요구되지 않는다. 왜냐하면 인수함수의 형변환이 그 변환을 처리하기때문이다. 그러나 형변환식을 리용하는것은 좋은 프로그람실습이다. 이렇게 함으로써 다른 프로그람작성자들은 변환이 일어 나고 있다는것과 이 변환이 의도하는것을 명확히 알수 있다.

형변환식의 다른 일반적인 리용은 옹근수더하기에 의한 잘라내기결과를 막는것이다. 야구공타격평균회수를 계산하려 하는 경우 평균회수는 야구공타격회수를 야구경기회전수로 나누어 결정한다. 다음의 코드는 정확한 결과를 내지 못한다.

```
int AtBats = 3;
int NbrOfHits = 1 ;
float BattingAverage = NbrOfHits / AtBats ;
```

문제는 나누기결과가 옹근수로 잘리우고 타률은 0.0으로 설정된다. 한가지 해결대책은 타수나 타격수중 어느 한 객체를 류점수형으로 정의하는것이다. 그러나 회전수나 타격회수는 소수가 될수 없으므로 그 대책은 인위적이다. 가장 좋은 해결책은 형변환식을 리용하는것이다. 나누기연산수중 어느 하나를 float형으로 형변환하여 류점수나누기를 진행할수 있다.

```
float BattingAverage = ((float) NumbrOfHits) / AtBats ;
```

이 명령문은 정확히 타률을 0.333으로 계산한다. 이밖에 형변환식이 코드작성을 명백히 하게 하는 다른 실례들도 있다. 이에 대해서는 필요할 때마다 지적할것이다. 이 절의 마감으로 란수를 리용하여 때때로 우연적인 선택을 진행하는 응용프로그람에 매우 쓸모 있는 2개의 함수들을 개발하자. 하나는 응용프로그람이 실행될 때마다 새로운 란수렬이 발생되도록 란수초기값을 설정하는 함수이다. 다른 함수는 규정된 간격으로 란수를 만든다. 여러가지 프로그람작성실례에서 이러한 함수들을 리용한다. 후에 사용자가 정의한 클라스에서 란수를 만들수 있는 능력을 일반화한다. 목록 5-1에 이런 편의함수의 머리부원형이 들어 있다.

목록 5-1.　　　　uniform.h 머리부파일

```
//란수편의함수의 대면부
#ifndef UNIFORM_H
#define UNIFORM_H
void InitializeSeed();
```

```
int Uniform(int Low, int High);
#endif
```

InitializeSeed()함수는 란수초기값을 설정한다. 프로그람 5-6과 같은 명령문을 리용하여 모조란수발생기의 초기값을 설정한다. Uniform()함수의 목적은 지정된 간격으로 모조란수를 만드는것이다. 그 간격은 2개의 파라메터 Low와 High를 경유하는 함수로 규정된다. 이름 Uniform은 만들어 진 수가 지정된 간격으로 균일하게 선택될것이라는 의미이다. 즉 그 간격의 매 개수는 균일하게 생성된다. 목록 5-2에 2개 함수의 실현이 있다.

목록 5-2. 파일 uniform.c에서 InitializeSeed( )와 Uniform 함수의 실현

```
//InitializeSeed함수와 Uniform함수의 실현
#include <iostream>
#include <string>
#include <stdlib.h>
#include <time.h>
#include "uniform.h"
using namespace std;
//InitializeSeed(): 란수발생기를 설정한다.
void InitializeSeed() {
    Srand((unsigned int) time(0));
}
//Uniform(): 아래준위와 웃준위사이의 값을 발생한다.
int Uniform(int Low, int High) {
    if (Low > High) {
        cerr << "Illegal range passed to Uniform" << end1;
        exit(1);
    }
    else {
        int IntervalSize = High ? Low + 1;
        int RandomOffset = rand() % intervalSize;
        return Low + RandomOffset;
    }
}
```

InitializeSeed()의 정의는 간단하다. 프로그람 5-6에 있는 main()함수에서와 같은 명령문을 리용하여 모조란수발생기초기값을 설정한다.

Uniform()함수는 좀 더 복잡하다. 우선 간격이 유효하게 지정되였는가 즉 Low가 high보다 크지 않는가를 확인한다. Low가 high보다 크면 오유통보가 생기고 함수가 오유신호를 하면서 완료된다. 간격이 적당하면 지정된 간격으로 옹근수값이 계산되여 intervalSize에 할당된다.

int IntervalSize = High - Low + 1 ;

다음 rand()에 의하여 생긴 모조란수를 IntervalSize로 나눈 나머지가 RandomOffset에 값주기된다.

int RandomOffset = rand() % IntervalSize ;

식 rand()% IntervalSize는 0부터 IntervalSize-1 까지 동일 한 수법으로 모조란수를 생성한다.

RandomOffset의 값이 0부터 IntervalSize-1까지의 모조란수이므로 Low+RandomOffset의 값은 간격 Low부터 Low+IntervalSize-1까지이다. IntervalSize가 High - Low + 1이므로 식 Low IntervalSize -1은 High와 같다. 따라서 Uniform이 되돌리는 값은 간격Low와 High사이에 있는 모조란수이다.

5.8 assert서고

assert서고는 프로그람개발시 유용한 전처리마크로 assert를 제공한다. assert마크로는 한개의 파라메터로써 완전한 식을 예견한다. assert마크로가 호출될 때 파라메터식이 령인가 알기 위하여 검사된다. 식이 령이 아니면 프로그람은 표준방식으로 계속 진행된다. 식이 령이면 프로그람은 프로그람을 끝내게 한 식, 원천파일이름,실패한 assert식의 그 파일행을 현시하는 표준오유흐름에 통보를 내보낸다. 정보를 현시한 다음 프로그람을 끝낸다. assert서고는 math서고와 마찬가지로 C언어를 기초로 한 서고이다. 따라서 프로그람에서 이 서고를 포함할 때 cassert를 리용한다.

프로그람 5-7은 assert명령문을 리용하여 그것이 처리하는 분모가 령이 아닌가를 확인한다. 프로그람이 파일 my.cpp에 포함된다고 하자. 그의 객체통보에서 령이 추출되면 assert마크로는 령을 시험하고 프로그람은 다음에 보여 주는것과 같은 오유통보를 현시하고 끝낸다.

Assertion failed: Denominator, file my.cpp , line 12

Assertion실패정보는 프로그람작성자에게는 유용할지라도 사용자에게는 의의가 없다. 따라서 assert마크로의 사용은 입력유효성이나 사용자위주의 오유통보를 대신할수 없다. 프로그람작성자는 보통 프로그람이 론리적으로 정확한가를 검사하기 위하여 오유수정시 assert마크로를 사용한다.

```
// 프로그람 5-7: 수입력의 상과 나머지계산
#include <casser>
#include <iostream>
#include <string>
using namespace std;
int main() {
    int Numerator;
    cout << "Enter numerator:";
    cin >> Numerator;
    int Denominator;
    cout << "Enter denominator:";
    cin >> Denominator;
    assert(Denominator); //
```

```
    int Ratio = Numerator / denominator;
    int Remainder = Numerator % Denominator;
    cout << Numerator << "/" << Denominator << " = "
      << Ratio << "with remainder" << Remainder << end1;
    return 0;
}
```

프로그람 5-7. 분모가 령임을 알리는 프로그람

경험

assert호출의 평가를 조종

assert호출에서 식이 평가되려면 마크로 NDEBUG가 정의되지 말아야 한다. 따라서 assert호출의 기정동작은 주장들을 평가하는것이다. 다음의 전처리기명령문은 다음에 오는 주장을 평가할 수 없게 한다.

```
#define NDEBUG
```

어떤 실현에서는 아래의 전처리기명령문을 리용하여 선언을 다시 평가할수 있다.

```
#undef NDEBUG
```

이러한 처리는 assert에 의한 동작이 NDEBUG가 정의되여 있는가를 검사하는 조건평가지령안에서 겹싸여 있기때문에 작용한다. 프로그람의 시작에 사용자에게 배포되는 NDEBUG정의를 포함하는것이 보통이다.

문제

7. 다음의 명령의 기능을 설명하시오.

```
using namespace std ;
```

8. iostream용namespace지령이 프로그람모듈에서 리용되지 않았으며 프로그람작성자가 cout흐름에 "Fatal Error"라는 문자렬을 쓰도록 하는 C++명령문을 쓰시오.
9. 오유통보의 현시와 기록에 리용되는 2개의 흐름의 이름은 무엇인가?
10. cout흐름에 int형객체 Head Count를 8진수로 삽입하는 C++명령문을 작성하시오.
11. oct, dec, hex조작자가 지속적이라고 하였는데 이것은 무엇을 의미하는가?
12. 다음의 코드를 실행하면 어떤 결과가 현시되는가?

```
int i = 16 ;
int j = 25 ;
int k = 56 ;
cout << hex << j << endl ;
cout << oct << j << "" << dec << k << endl ;
```

13. 다음의 코드의 실행결과는 무엇인가?

```
cout << "123456789" << endl ;
cout << setw(10) << "Hello" << endl ;
cout << "Good bye " << endl ;
```

14. 다음의 코드의 실행결과는 무엇인가?

```
int k = 3 ;
cout << setfield('$') << setw(5) << k << endl ;
```

15. 다음의 코드의 실행결과는 무엇인가?

```
cout <<"123456789 " << endl ;
cout << "$" << left << setw(5) << "z" << "$" << endl;
cout << "$" << left << Setw(4) << "z" << "$" << endl;
```

16. 다음의 코드의 실행결과는 무엇인가?

```
cout << "123456789" << endl ;
cout << "$" << right << setw(5)<< "z" << "$" << endl;
cout << "$" << right << Setw(4) << "z" << "$" << endl;
```

17. 다음의 코드의 실행결과는 무엇인가?

```
bool Yes = true;
cout << noboolalpha << Yes << endl;
```

18. 다음의 코드의 실행결과는 무엇인가?

```
int i = 16;
int j = 25;
int k = 56;
cout << setbase(16) << i << endl;
cout << setbase(8) << j << " " <<setbase(10) << k << endl;
```

19. 다음의 코드의 실행결과는 무엇인가?

```
int i = 16;
cout << showpos << i << endl;
```

20. 다음의 코드의 실행결과는 무엇인가?

```
int i = 16;
cout << oct << showpos << i << endl;
```

21. 모조란수옹근수를 되돌리는 stdlib함수의 이름은 무엇인가?
22. 란수발생기의 초기값을 설정하는 stdlib함수의 이름은 무엇인가?
23. 쇠돈을 1000번 던져 앞판이 나오는 빈도수를 구하는 프로그람을 작성하시오.
24. data.txt라는 파일에서 옹근수를 읽어 들이는 프로그람을 작성하시오. 프로그람은 파일에서 홀수개수를 출력한다. 출력은 다음과 같이 한다.

```
There are 25 odd number(s) in the file data.txt
```

25. int형객체 Count가 0이면 오유를 알리는 명령문을 마크로를 리용하여 작성하시오.
26. int형함수 Index가 부수를 되돌리면 오유를 내는 마크로명령을 작성하시오.

5.9 알아 둘 점

- ✓ 쏘프트웨어의 재리용은 프로그람을 빨리 개발할수 있고 효과적으로 동작할수 있게 한다. 쏘프트웨어의 기본원천의 하나는 표준서고이다.
- ✓ C++는 여러가지 기능을 수행하는 많은 서고를 제공하고 있다. 일부 중요한 흐름서고들은 iostream, iomanip, fstream서고들이다. 이 서고들은 조종된 방식으로 정보를 삽입하고 추출해내는것과 같은 동작을 제공한다.
- ✓ 머리부파일은 함수대면부, 상수, 변수정의, 그리고 클라스서술의 집합체이다. 머리부파일은 include지령을 통해 프로그람에 병합된다.
- ✓ 함수에 대한 정보는 파라메터형식으로 넘어 간다. 함수의 계산은 보통 되돌림값으로 된다. 함수에 의해 되돌려 진 값의 형은 복귀형이다. 값을 되돌리지 않는 함수는 void형을 가진다.
- ✓ 호출시 파라메터들을 실제파라메터라고 한다. 실제파라메터는 호출된 함수에서 형식파라메터에 의하여 표현된다.
- ✓ 함수가 호출될 때 조종흐름은 함수를 호출하는것으로부터 호출된 함수에로 넘어 간다. 호출된 함수가 과제를 수행할 때 조종이 다시 함수를 호출한쪽으로 넘어 간다. 호출된 함수가 되돌림값을 가지면, 그 값은 본질에 있어서 호출을 대신한다.
- ✓ 매 함수호출은 함수의 활성화레코드를 창조한다. 형식파라메터와 함수에서 정의된 다른 객체들의 값이 활성화레코드에 보관된다.
- ✓ 함수는 호출전에 선언되거나 정의되여야 한다. 원형선언은 함수의 대부분을 서술한것이다. 함수정의에는 함수대면부와 그의 본체가 다 포함된다. 대면부서술은 복귀형, 함수이름, 파라메터목록의 형태를 규정한다.
- ✓ 실제파라메터를 넘기는 한가지 방법은 값파라메터에 의한 넘기기방식이다. 실제파라메터가 이 방식으로 넘어 갈 때 형식파라메터는 그것이 실제파라메터값으로 초기화되므로 값파라메터라고 한다. 형식파라메터에 대한 그이후의 변화는 실제파라메터에 영향을 주지 않는다.
- ✓ 모든 표준서고들은 그 서고들에서 정의된 함수들을 원형선언하는 머리부파일들을 가진다.
- ✓ 프로그람에 표준서고를 포함할 때 보통 명령문

  ```
  using namespace std;
  ```

  를 리용한다. 이 명령문이 없다면 서고에 대하여 매번 참조할 때마다 std ::를 앞에 붙여야 한다.
- ✓ 전처리기는 프로그람파일에서 3가지 종류의 지령처리를 할 책임이 있다. 파일포함지령은 콤파일되여야 하는 번역단위의 한 부분으로 될 파일을 규정한다. 마크로정의와 호출은 본문을 대신한다. 조건부콤파일지령은 프로그람작성자가 콤파일될 코드행을 제한할수 있게 한다.
- ✓ 입력과 출력흐름을 표현하기 위하여 클라스계층이 존재한다. 그 계층의 뿌리는 ios_base클라스이다.
- ✓ \iostream과 ostream은 중요한 흐름클라스들이다. istream클라스는 입력흐름보조계층의 뿌리이며 ostream클라스는 출력흐름보조계층의 뿌리이다.
- ✓ ifstream과 ofstream클라스는 파일을 조작하는 흐름을 정의한다. 이 흐름들은 추출과 삽입처리를 진행한다.
- ✓ iostream과 fstream클라스는 추출과 삽입을 다 할수 있는 흐름을 정의한다.
- ✓ iostream 서고는 두가지 오유통보흐름을 정의한다. Cerr 흐름은 오유통보를 완충하지 않으며 clog흐

름은 오유통보를 완충한다.

- ✓ iostream서고는 또한 추출과 삽입흐름을 진행하는 방식을 규정하는 조작자를 정의한다.
- ✓ 련속 I/O조작을 조종하는 조작자는 지속적이다.
- ✓ iostream조종인 dec, oct, hex는 수자들이 현시되는 밀수를 조종한다.
- ✓ iostream조작자 flush는 출력완충기의 내용이 즉시에 현시되도록 한다.
- ✓ iomanip서고 역시 흐름조작자를 제공한다. 그중 주요한 두 조작자가 setw()와 setprecision()인데 이것들은 삽입폭과 정확도를 지정한다.
- ✓ iostream조작자 fixed와 scientific는 수값이 십진법이나 과학적표기법으로 현시되도록 한다.
- ✓ iostream 조작자 skipws와 noskipws는 공백이 삽입조작자사용시 추출가능한가를 조종한다.
- ✓ assert서고는 식을 평가할수 있는 assert마크로를 정의한다. 식이 거짓이면 프로그람은 중지된다. 식이 참이면 프로그람은 계속 진행된다. assert서고는 C언어에 기초한 서고이다. 서고를 포함시키려면 머리부파일 cassert나 assert.h를 리용한다.
- ✓ stdlib서고는 잡다한 편의함수의 집합체이다. 서고를 포함시키려면 cstdlib.h머리부파일이나 stdlib.h머리부파일을 리용한다.
- ✓ stdlib서고함수 exit()는 호출될 때 프로그람을 즉시에 끝낸다. exit()의 파라메터는 프로그람되돌림값으로 리용된다. stdlib서고는 RAND_MAX를 통해 구간 0에서 모조함수를 되돌리는 함수 rand()을 제공한다. 여기서 RAND_MAX는 stdlib상수이다.
- ✓ stdlib서고는 rand()를 리용하여 얻을수 있는 란수렬을 변경하기 위한 srand()함수를 제공한다. 함수호출 Srand((unsigned int)time(o))은 호출할 때마다 각이한 란수렬을 초기화하여야 한다.
- ✓ 형변환식은 한 자료형을 다른 자료형으로 명백히 변환한다.
- ✓ 형변환식은 프로그람작성자가 옹근수나누기보다는 류점수나누기 등 특수한 연산을 진행하려 하는 경우 유용한다.
- ✓ 형변환식은 서고함수의 되돌림값을 적당한 자료형으로 변환하는데 유용하다.

참고할 책

이 장에서는 서술한 여러가지 서고의 함수들과 원소들은 다음의 책에서 참고하였다.

- B.Stroustrup, The C++ Programming Language: Third Edition, Reading, MA: Addison-Wesley, 1998
- X3 Secretariat, Draft Standard ? The C++ Language, X3J16/97-14882, Washington, DC: Information Technology Council(NSITC), 1997.
- P.J.Plauger, The Standard C Library, Englewood Cliffs, NJ: Prentice-Hall, 1992.

련습문제

5.1 콤퓨터지원전화호출을 쉽게 하는 서고를 창조하여야 한다고 가정하자. 의뢰기응용프로그람을 지원하는 서고에 어떤 함수가 있어야 하는가?

5.2 콤퓨터지원도서검사관리를 쉽게 하는 서고를 창조하여야 한다고 가정하자. 의뢰기응용프로그람을 지원하는 서고에 어떤 함수가 있어야 하는가?

5.3 전화술이나 도서검사관리외에 콤퓨터지원이 적합한 응용분야의 두가지를 지정하시오.

5.4 체계에서 머리부파일이 보관되여 있는 등록부를 찾으시오.
5.5 조종흐름을 다시 토론하시오.
5.6 값에 의한 넘기기가 아닌 파라메터넘기기방식들에 대하여 생각해 보시오. 그들이 어떻게 효과적인가 토론하시오.
5.7 거의 모든 조작체계들이 파일을 문자로서 지정한다. 이 문자는 iostream int형상수 EOF와 다르다. 이 문자가 표준입력에 도착하면 사용자는 더이상 입력할 값이 없다는것을 나타낸다. 대부분의 개인용콤퓨터에 기초한 체계에서는 파일끝문자가 ctrl+z이다. 다른 체계들에서는 ctrl+d이다. 현재 사용하고 있는 체계에서 파일끝문자를 결정하시오. 그리고 그것을 결정하는 방법을 쓰시오.
5.8 cout의 출력요구가 실패하였다고 가정하자. 즉 령이 귀환되였다. 이런 사건은 무엇때문인가? 프로그람사용자에게 알리려면 어떻게 할수 있는가?
5.9 2중정확도류점수값 x를 파라메터로, 같은 자료형은 복귀형으로 하는 QuarticRoot()함수를 원형선언하시오.
5.10 2개의 옹근수값 width와 height를 가지며 옹근수값을 되돌리는 함수 Area()를 원형선언하시오.
5.11 문자값 C를 가지며 론리값을 돌려 주는 함수 IsMathSymbol()을 원형선언하시오.
5.12 파라메터도 전혀 없고 되돌림값이 옹근수형인 함수 GetNumber()를 원형선언하시오.
5.13 지정된 구의 체적을 돌리는 함수를 원형선언하시오.
5.14 제정된 시간동안에 제정된 속도로 가속하는 객체가 멈추기 시작할 때의 속도를 되돌리는 함수를 원형선언하시오.
5.15 섭씨온도를 화씨온도로 변환하는 함수를 원형선언하시오.
5.16 표준출력흐름 cout에 5개의 공백을 현시하는 함수를 원형선언하시오.
5.17 표준출력흐름 cout에 규정된 수의 공백을 현시하는 함수를 원형선언하시오.
5.18 의미가 유지되는 조작자란 무엇인가? setw()가 어째서 의미가 유지되지 않게 설계되였는가를 생각해 보시오.
5.19 iostream서고의 실행이 류점수를 10진법으로 현시하는가 아니면 파학적표기법을 리용하겠는가를 결정하시오.
5.20 프로그람 5-1을 수정하여 2차방정식의 실수풀이와 a가 0일 때의 결과처리를 진행하시오.
5.21 프로그람 5-1에서 자료를 파일흐름에 얻으면 좋은가? 리유는 무엇인가?
5.22 프로그람 5-3을 수정하여 사용자가 지정한 파일에서 입력하도록 하시오.
5.23 사용자에게 자료입력상태를 알리는 프로그람에서 cout흐름을 리용하는가 cerr흐름을 리용하는가? 그 리유를 밝히시오.
5.24 다음의 프로그람은 어떤 결과를 내는가? 리유는 무엇인가?

```
int main() {
    cout << "12345678901234567890" << endl;
    cout << setw(5) << 100 << endl;
    cout << 100 << endl;
    cout << setprecision(4) << 71.498000000001 << endl;
    cout << setfill(ioi);
    cout << setw(10) << 88 << endl;
    cout << setw(20) << "Hello World" << endl;
    return 0;
}
```

5.25 다음의 공식을 C++식으로 쓰시오. 부록 2에로 math서고함수를 리용하시오.

a) $(\sin x)^2 \times (\cos x)^2$

b) $e^{\frac{1}{2}\sqrt{tna \cos x}}$

c) $$\frac{\left\lfloor \log\left(\frac{x^2}{1-x}\right) \right\rfloor}{\left\lceil x^{5+x} \right\rceil}$$

5.26 다음의 문자렬을 표준출력에 현시하는 프로그람을 작성하시오. 한행에 한 문자씩 배치하되 중심에 놓아야 한다.

Greetings each inhabitants
Take me to your leader
Is that am there is
It is not out fault

5.27 표준입력으로부터 한 문자를 추출하여 표준출력에 표형식으로 삽입하시오. 표안에 있는 정보는 파라메터로 추출문자를 호출할 때 ctype서고의 여러가지 함수들(부록 2 참고)에서 얻은 값이다.
5.28 문제 5.27에서 출력이 파일 ctype.tbl에 삽입되도록 재작성하시오.
5.29 5개의 입력값을 추출하는 순환없이 입력하는 프로그람을 작성하시오. 프로그람은 이 값들을 별표를 리용하여 기둥도표로서 현시한다. 실례로 입력이

5 9 4 12 7

이라면

```
5 :  *****
9 :  *********
4 :  ****
12:  ************
7 :  *******
```

이다. 풀이방향: iostream를 리용하여 채우기문자를 수정하시오.
5.30 문제 5.29의 풀이를 다시 작성하여 setw()가 흐름조작자만 리용되게(순환은 허용되지만 문자렬은 사용하지 않는다.) 처리하시오.
5.31 사용자에게 파일이름과 문자렬을 입력할데 대한 통보를 내보내는 프로그람을 작성하고 수행하시오. 이 프로그람은 그다음 지정된 파일에서 문자의 출현회수를 계산하여야 한다.
5.32 사용자가 3각형의 세변 a, b, c를 입력하고 3각형생성조건이 맞으면 3각형의 면적을 현시하는 프로그람을 작성하시오.
5.33 사용자로부터 시작량과 해수를 입력하여 생산량이 2배 늘어 나도록 퍼센트비률을 결정하는 프로그람을 작성하시오.
5.34 사용자로부터 시작량과 증가퍼센트비률을 입력하여 몇해정도 걸려야 생산량이 2배로 되는가를 결정하는 프로그람을 작성하시오.
5.35 문제 5.33과 5.34에서 시작량을 왜 입력하는가?
5.36 표준입력흐름 cin의 문자들은 무시될수 있는 수자문서들은 제외하고 표준출력흐름 cout로 복사하는 프로그람을 작성하시오.
5.37 표준입력흐름 cin의 문자들을 수자문자들을 제외하고 표준출력흐름 cout로 복사하는 프로그람을 작성하시오. 수자문자는 대신 수자이름을 현시한다. 즉 0은 zero, 1은 one 등
5.38 cin에서 입력한 문자들을 출력하되 모든 공백문자들을 해당한 확장문자렬로 고쳐 출력하시오. 실례로 타브가 입력되면 문자 \t를 삽입한다.
5.39 Simple Window의 임의의 위치에 청색Rectangle shapes를 그리는 프로그람을 작성하시오. Rectangleshapes는 우연적인 크기를 가진다.
5.40 Simple Window의 임의의 위치에 청색RectangleShapes를 그리는 프로그람을 작성하시오. RectangleShapes는 우연적인 크기와 색갈을 가진다.

제 6 장. 프로그람작성자정의함수

소 개

제5장에서는 서고함수를 리용하여 과제를 수행하는 방법에 대하여 보았다. 그러나 의의가 큰 쏘프트웨어대상과제는 그외에도 새로운 함수를 설계하고 실현할것을 요구한다. 이 장에서는 값파라메터를 가진 프로그람작성자정의함수의 기초를 고찰한다. 우리의 고찰은 호출과 파라메터, 국부적 및 전역적범위의 토론을 포괄한다.

기본개념

- 사용자정의함수
- 호출과 조종흐름
- 파라메터
- 원형선언
- 활성화레코드
- return명령문
- 국부객체
- 유효범위
- 전역객체
- 이름의 재리용
- 실행부파일
- 머리부파일
- 표준클라스 ostringstream
- 표준클라스 istringstream
- Label클라스
- 표준본보기서고 (Standard Template Library : STL)
- 참조파라메터
- 상수파라메터
- 기정파라메터
- 함수다중정의
- 함수다중정의호출
- 부작용
- 재귀처리

6.1 기초

현재 존재하는 쏘프트웨어설계를 리용하여 프로그람작성과제를 수행하는것은 무엇보다 중요한 문제풀이전략이다. 이것은 무엇보다도 서고리용이 시간과 자금을 절약하는데 아주 효과적이기때문이다. 그러나 대다수의 쏘프트웨어과제에 대하여 서고함수들만을 리용할수는 없다. 또한 사용자가 정의한 함수를 창조하는것이 필요하다.

자체의 함수를 설계하는 경우 정확성과 효과성도 중요하지만 쏘프트웨어의 재리용법도 아주 중요하다. 재리용성은 정비와 앞으로의 대상과제에 드는 비용이 최소로 되게 한다.

이 장에서 설계하는 함수는 전역객체를 리용하는 한개 프로그람을 제외하고는 다음의 두가지 특성을 가진다. 함수에 의하여 리용될 외부정보는 함수의 값파라메터목록이나 입력흐름으로부터의 추출로부터 얻어 진다. 함수로부터 통신되는 정보는 함수의 되돌림값 혹은 출력흐름에로의 삽입에 의하여 보내여진다.

정보흐름에 대한 이러한 조종은 함수를 쉽게 실현하고 리해할수 있도록 한다. 이 정보흐름을 그림 6-1에 주었다.

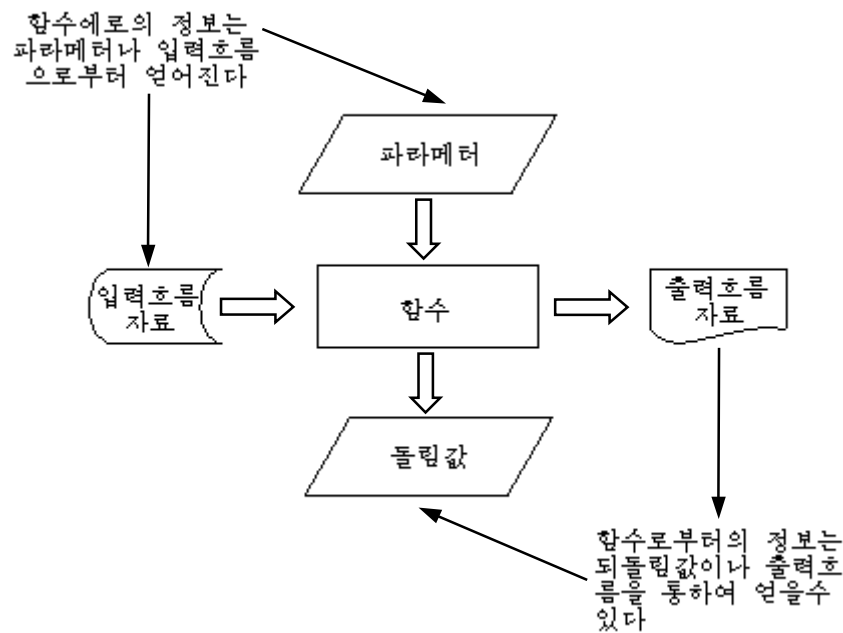



그림 6-1. 값파라메터를 리용한 함수에로의 정보흐름모형

6.1.1 함수정의문장론

함수정의는 대면부와 그의 동작을 포함하는 명령목록의 서술을 포함한다.

C++에서는 여러 함수들이 같은 이름을 가지고 있으므로 함수이름만을 주는것은 명백한 정의로 되지 않는다. 대신 C++는 형식파라메터의 이름과 형을 포함하는 대면부전체를 리용하여 어느 함수가 정의되였는가를 지적한다.

프로그람 6-1은 함수 CircleArea()의 정의를 포함한다. 이 함수는 원의 면적을 계산하는데 그 프로그람의 main()함수에서 호출된다. CircleArea()함수는 단일**float**형파라메터 r와 **float**형되돌림값을 가진다.

```
// 프로그람 6-1: 원의 면적을 계산하기
#include <iostream>
#include <string>
using namespace std;
//CircleArea():반경 r를 가진 원의 면적을 계산한다.
float CircleArea(float r) {
    const float Pi = 3.1415;
    return Pi *r *r;
}
// main(): 사용자가 정의한 원의 면적값에 대한 계산과 현시를 관리한다.
int main() {
    // 지령대기문에서 반경값을 입력한다. 반경의 입력을 독촉하고 그것을 읽는다.
```

```
    float DesiredRadius;
    cout << "Circle radius (number):";
    cin >> DesiredRadius;
    // 면적을 계산한다.
    float Area = CircleArea(DesiredRadius);
    cout << "Area of circle with radius"
      <<DesiredRadius << "is" << Area << endl;
    // 프로그람이 완료되였다.
    return 0;
}
```

프로그람 6-1. main()에서 사용자정의함수호출

main()함수를 리용하는 이전 실례에서처럼 함수의 동작은 함수본체에 주어 진다. 함수본체는 왼쪽과 오른쪽대괄호안에 있는 명령문목록이다.

프로그람 6-1의 정의로부터 CircleArea()함수의 본체는

```
const float Pi = 3.1415;
return Pi *r *r;
```

이다. 함수본체에서 리용되는 명령문에는 제한이 없다. main()함수에서 쓰이던 명령문이면 다른 함수에서도 쓸수 있다. 첫번째 명령문은 상수 π에 대한 근사 Pi를 정의한다. 다음명령문은 함수의 **return**명령문이다. 함수는 상수 Pi와 원면적공식 πr^2 을 리용하여 파라메터 r를 반경으로 하는 원의 면적식을 **float**에 대응하여 계산한다.

함수는 **return**명령문을 리용하여 과제가 다 완료되였음을 지적한다. 함수되돌림형이 **void**가 아니면 그 명령문은 또한 함수의 되돌림값을 함수호출자에 전달한다. **return**명령문에 의해 함수의 결과값이 함수호출자에게 넘어 간다. **return**명령문의 형식은 다음과 같다.

여기서 되돌림식 ReturnExpression은 함수대면부에 지정된 되돌림형과 같은 형이여야 하는 값으로 평가한다(같지 않은 경우는 ReturnExpression을 자동적으로 되돌림형으로 변환하려고 한다). CircleArea()함수에서 되돌림값은 **float**식 Pi*r*r의 값이다. 일단 되돌이식이 계산되면 되돌이식값을 리용하여 호출시 실행을 계속한다.

비록 필요없지만 **void**형함수는 **return**명령문을 리용할수 있다. 만일 **void**형함수에 **return**명령문을 쓰게 되면 함수가 끝난 다음 조종흐름이 호출한 함수에로 되돌아 간다는것을 명백히 가리킨다. **void**형함수의 **return**명령문에는 식이 허용되지 않는다.

6.1.2 호출과 조종흐름

서고함수호출과 마찬가지로 호출시 실제파라메터와 프로그람작성자정의함수의 형식파라메터사이의 일치성은 파라메터의 상대위치에 의하여 결정된다. 첫번째 실제파라메터는 첫번째 형식파라메터와 련결되며 두번째 실제파라메터는 두번째 형식파라메터와 련결되여야 한다.

프로그람 6-1에 있는 main()함수에서 그의 활성화레코드에는 DesiredRadius와 Area객체가 있다. DesiredRadius에 추출된 값이 1, 0이고main()의 실행에 의하여 Area의 정의를 곧 실행하려 한다고 하자(의문부호는 프로그람이 아직 그 객체의 값을 설정하지 않았다는것을 가리킨다).

main()	
DesiredRadius	10,000
Area	?

CircleArea()함수가 Area초기화를 위하여 호출될 때, 거기에 활성화레코드가 또 창조된다. 그 활성화레코드에는 형식파라메터 r가 실제파라메터의 값으로 초기화되여 있는데 이 경우에는 그 값이 DesiredRadius의 값이다. 이 호출에 대한 활성화레코드는 아래와 같다.

CircleArea()	
r	10.0000
Pi	3.1415

형식파라메터 r가 값파라메터이므로 DesiredRadius가 r를 초기화하는데 일단 리용된다면 DesiredRadius와 r는 서로 독립으로 된다. R가 CircleArea()에서 리용될 때마다 r의 값은 함수의 현재활성화레코드에 기억된다. CircleArea()가 r를 변화시키면 CircleArea()의 현재활성화레코드안에서 변화가 생기며 DesiredRadius에는 영향이 없다. 형식파라메터 r는 함수 CircleArea()에만 국한되는 국부적인 객체이다. 파라메터는CircleArea()호출시에 창조되고 CircleArea()함수가 완성되여 그의 활성화레코드기억기를 해방할 때 해체된다.

실례의 CircleArea()호출에서 되돌림값은 314.15이고 이 값을 써서 Area을 초기화한다. 이때 main()의 활성화레코드는 다음과 같이 된다.

main()	
DesiredRadius	10,000
Area	314.15

5장에서 살펴 본것처럼 호출중의 함수가 원형화 혹은 정의될 때까지 어떤 호출도 나타낼수 없다. 프로그람 6-1에서는 main()함수를 정의하기전에 CircleArea()함수를 정의한다. 이 순서화는 main()이 CircleArea()를 호출하게 한다. 비록 함수정의는 CircleArea()가 먼저 되여 있어도 프로그람실행은 main()에서부터 시작된다.

6.2 흥미 있는 문제

가락지빵의 체적을 계산한다고 가정해 보자. 여기서 가락지빵이란 속대(원기둥심)를 뺀 원기둥을 말한다. 이러한 형식을 그림 6-2에 주었다.

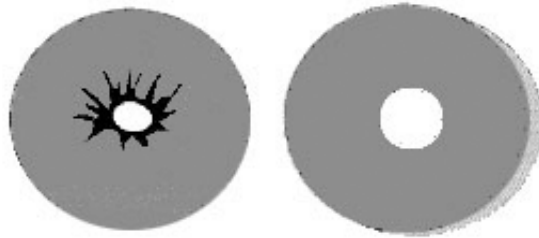



그림 6-2. 가락지빵은 원기둥모양의 속대를 뺀 원기둥으로 근사화할수 있다

원기둥의 체적(크기)은 반경이 r이고 높이가 h일 때 공식 $\pi r^2 h$로 구한다. 원의 면적을 계산하던 경험에 기초하여 원기둥의 체적을 간단히 구할수 있다. CylinderVolume()함수를 리용하기로 한다.

경험

상수값파라메터

이 장에서 정의하는 함수들은 형식파라메터를 상수(실제파라메터를 리용하여 초기화되는)로서 리용한다. 형식파라메터들이 이런 방법으로 리용되기때문에 그것들은 const값파라메터로 선언할수 있을것이다. 실례로 다음의 코드에서는 **const**파라메터선언을 리용하는 CircleArea()함수를 정의한다.

```
float CircleArea(const float r) {
    const float Pi = 3.1415;
    return Pi *r *r;
}
```

객체가 변경되지 말아야 하는 경우 콤파일러나 의뢰기에 여분의 정보를 제공하기 위하여 **const** 수식자를 적용하였다. 그러나 여기서는 필요로 되지 않는 함수실현에 대하여 의뢰기에 정보를 주고 있다. 즉 의뢰기는 함수가 자기 일감을 수행하는가 실제파라메터가 변경되지 않는가에만 관심을 두고 있다. 형식파라메터를 값파라메터로 하면 보통 의뢰기의 관심사를 충분히 처리할수 있다.

```
//CylinderVolume(): 반경은 r이고 높이는 h인 원기둥의 체적계산
float CylinderVolume (float r, float h) {
    const float Pi = 3.1415;
    return Pi *r *r *h;
}
```

가락지빵의 체적은 2개의 원기둥체적의 차로 계산할수 있다. DonutSize()함수가 이 기능을 수행한다. 이 함수는 가락지빵의 두께에 따라 2개의 해당한 원기둥의 반경을 파라메터로 받아 두 원기둥의 체적의 차를 되돌린다.

```
//DonutSize():중심으로부터 바깥걸면까지의 반경 Outer, 아낙걸면까지의 반경 Inner,
//높이 Width인 속이 빈 원기둥의 체적을 계산한다
float DonutSize(float Outer, float Inner, float Width) {
    float OuterSize = CylinderVolume(Outer, Width);
    float HoleSize = CylinderVolume(Inner, Width);
```

```
        return OuterSize - HoleSize;
    }
```

프로그람 6-1에 있는 CircleArea()의 호출이 파라메터 r를 기억하도록 활성화레코드용기억기가 따로 설정될것을 요구하는것과 마찬가지로 DonutSize()와 CylinderVolume()함수의 호출은 그것들이 사용하는 파라메터와 객체를 기억하도록 활성화레코드용기억기를 따로 설정할것을 요구한다. DonutSize() 파라메터가 초기화될 때 파라메터 Outer와 Inner는 초기화에 리용된 식과 객체와 독립으로 된다. 앞서 이야기하였지만 이 독립성은 값파라메터넘기기의 특성이다. DonutSize()안에서는 **float**객체로서 Outer와 Inner 그리고 Width를 마음대로 사용할수 있다. 특히 그것들을 함수호출에 리용할수 있다. 따라서 OuterSize의 정의에서는 CylinderVolume()함수가 처음 호출될 때 파라메터 r와 h를 초기화하는데 Outer와 Width를 사용하여도 문제가 생기지 않는다. 마찬가지로 CylinderVolume()을 두번째로 호출할 때에도 Inner와 Width를 파라메터초기값으로 리용하여도 된다.

DonutSize()가 CylinderVolume()함수를 호출하려면 그 프로그람파일에서 이미 CylinderVolume()함수가 미리 정의되거나 원형화되여야 한다. 프로그람 6-2에 원기둥고리문제의 완전한 풀이를 주었는데 여기서CylinderVolume()는 DonutSize()프로그람에서 main()함수정의에 앞서 다른 함수들을 미리 선언하였으며 특히 CylinderVolume()함수는 DonutSize()함수가 정의되기전에 미리 선언되였다.

C++에서는 함수가 미리 선언되는 즉시 리용되므로 프로그람파일시작에 이러한 함수들의 미리 선언을 진행하고 DonutSize()와 CylinderVolume()를 임의의 순서로 정의할수 있다. CylinderVolume()의 원형화는 함수정의할 때 r와 h를 사용하는 파라메터 Radius와 Width를 준다. 이런 차이는 매우 정밀하다. 즉 원형선언은 함수의 대면부형식만을 서술한다. 제5장을 다시 상기시킨다면 식별자이름을 지원하는 것은 원형선언에서 선택적이다. 즉 형식파라메터는 형선언만 하여도 충분하다. 그러나 의미를 가지는 이름은 프로그람작성자들에게 코드리해를 쉽게 하여 준다.

쏘프트웨어개발에서 기본품은 현재 있는 코드를 수정하는데 드므로 코드를 리해하기 쉽게 하여야 한다. OuterEdge, InnerEdge, Thuckness에 각각 2.5, 0.5, 0.75를 입력하였다.

```
// 프로그람 6-2 : 사용자가 정의한 과자의 크기를 계산한다.
#include <iostream>
#include <string>
using namespace std;
//형선언
float DonutSize(float Outer, float Inner, float Width);
float CylinderVolume(float Radius, float Width);
//main():사용자가 정의한 과자의 크기를 계산하고 현시한다.
int main() {
    //과자의 크기를 입력한다.
    cout << "Outer edge donut radius:" ;
    float OuterEdge:
    cin >> OuterEdge;
    cout << "Hello radius:" ;
```

```
    float InnerEdge;
    cin >> InnerEdge;
    cout << "Donut thickness:" ;
    float Thickness;
    cin >> Thickness;
    //과자의 크기를 계산하고 현시한다.
    cout << endl << "Size of donut with" << endl;
      << "    radius" << OuterEdge << endl;
      << "    hole radius" <<InnerEdge << endl;
      << "    thickness" <<Thickness << endl;
      << "is "
      << DonutSize(OuterEdge, InnerEdge, Thickness)
      << endl;
    return 0;
}
//DonutSize():과자의 중심에서 바깥직경과 내부직경을 계산하고 과자의
//두께를 계산한다.
float DonutSize(float Outer, float Inner, float Width) {
    float OuterSize = CylinderVolume(Outer, Width);
    float HolesSize = CylinderVolume(Inner, Width);
    return OuterSize - HoleSize;
}
//CylinderVolume(): 반경 r와 높이 h를 가진 원통의 크기를 계산한다.
float CylinderVolume(float r, float h) {
    const float Pi = 3.1415;
    return  Pi *r *r *h;
}
```

프로그램 6-2. 사용자정의함수를 리용하여 과자크기계산

함수 main()에 대한 활성화레코드는 다음과 같다.

main()	
OuterEdge	2.5000
InnerEdge	0.5000
Thickness	0.7500

출력명령문에서 DonutSize()가 호출되면서 조종흐름이 DonutSize()의 복사에 림시 넘어 간다. 이때 파라메터 Outer, Inner, Width는 각각 2.5, 0.5, 0.75로 초기화된다. 파라메터가 초기화된 다음 DonutSize()함수가 호출되면 활성화레코드는 다음과 같이 된다.

main()	
OuterEdge	2.5000
InnerEdge	0.5000
Thickness	0.7500
OuterSize	?
HoleSize	?

Outer를 초기화함으로써 DonutSize()에서 CylinderVolume()의 복사본에로 조종을 림시 옮긴다. CylinderVolume()의 r와 h파라메터는 2.5와 0.75로 된다.

CylinderVolume()	
r	2.500
h	0.750

Pi를 정의한후 CylinderVolume()는 Pi*r*r의 값을 계산하고 되돌리는데 그 값은 14.7258이다. 조종이 DonutSize()에로 넘어 가고 CylinderVolume()의 활성화레코드기억기가 해방된다. 이때 DonutSize()의 활성화레코드는 다음과 같다.

DonutSize()	
Outer	2.5000
Inner	0.5000
Width	0.7500
OuterSize	14.7528
HoleSize	?

DonutSize()에서 Inner를 초기화할 때 다시 DonutSize()에서 CylinderVolume()의 복사본에로 조종이 림시이동한다. 이때 파라메터 r와 h의 복사본에로 조종이 림시이동한다. 그리고 파라메터 r와 h는 0.5, 0.75로 초기화된다.

CylinderVolume()	
r	0.500
h	0.750

Pi를 정의한후 함수는 Pi*r*r*h를 계산하여 되돌린다. 값은 대략 0.5890이다. 조종은 다시 DonutSize()에로 넘어 가고 CylinderVolume()활성화레코드기억기는 해방된다. 이때 DonutSize()의 활성화레코드는 다음과 같다.

DonutSize()	
Outer	2.5000
Inner	0.5000
Width	0.7500
OuterSize	14.7528
HoleSize	0.5890

그다음 DonutSize()의 **return**명령문이 실행된다. 되돌림식의 값은 14.1368이다. 되돌림될 때 활성화레코드기억기가 해방되고 조종이 main()에로 넘어 간다. 그리고 출력명령문에는 14.1368이라는 값이 넘어 간다. 프로그람 6-2의 실행결과는 아래와 같다.

```
Outer edge donut radius: 2.5
Hole radius: 0.5
Donut thickness: 0.75
Size of donut with
   radius 2.5
   hole radius 0.5
   thickness 0.75
is 14.1368
```

6.3 몇가지 쓸모 있는 함수들

프로그람에는 Max()와 Min()이 많이 리용된다. 이 두 함수는 다 **int**형값을 되돌리며 2개의 **int**형 파라메터를 가진다. Max()함수는 2개의 파라메터중 큰 값을 되돌리며 Min()함수는 작은 값을 되돌린다. Max()함수는 다음과 같이 정의할수 있다.

```
//Max(): 두개의 파라메터에서 큰 값을 결정한다.
int Max(int a, int b) {
    if (a < b)
        return b;
    else
        return a;
}
```

함수는 두 파라메터를 먼저 비교한다. 식 a<b가 true이면 b가 큰 수이므로 b를 되돌린다. 반대로 a<b가 false이면 a가 큰 수이므로 a를 되돌린다. 우의 실례는 한개 함수에 두개이상의 **return**명령을 가질수 있다는것을 보여 준다. Min()함수는 Max()함수와 비슷하다. 차이점은 비교식에서 크기기호 >가 쓰인것이다.

```
//Min():제일 작은 값을 결정한다.
int Min(int a, int b) {
    if (a > b)
        return b;
    else
        return a;
}
```

입력된 두 수가운데서 큰 수와 작은 수를 찾는 프로그람을 Max()와 Min()을 리용하여 작성한 실례를 아래에 주었다.

```
cout << "please enter a number (integer):" ;
int Value;
cin >> Value1;
cout << "please enter a number(integer): " ;
int Value2;
cin >> Value2;
cout << "MAX: " << Max(Value1, Value2) << endl;
cout << "Min: " << Min(Value1, Value2) << endl;
```

앞에서 주의를 준것처럼 실제파라메터와 형식파라메터는 이름이 같지 않아도 된다. 실제파라메터와 형식파라메터는 놓이는 위치의 순서가 같아야 한다. 위치가 같게 되면 함수는 계산을 진행하여 적당한 값을 되돌린다. 표준본보기서고(STL)의 알고리듬서고는 표준적으로 함수 Max()와 Min()을 제공하고 있다. 여기서 제공하는 함수이름은 max()와 min()이다(본보기함수는 14장에서 론의한다). 이 함수를 프로그람에서 사용하려면 include지령을 가진 다음의 명령문을 삽입하여야 한다.

```
#include <algorithm>
```

이밖에 많이 쓰는 함수는 int형함수 PromptAndGet()이다. 이 함수는 흐름 cout에 입력한 수를 현시한다. 그다음 입력흐름 cin으로부터 받은 값을 추출해 낸다. PromptAndGet()의 되돌림값은 추출해 낸 값으로 된다. 함수는 파라메터를 가지지 않는다.

```
//PromptAndGet():옹근수값을 추출해 낸다.
int PromptAndGet() {
    cout << "Please enter a number (integer): ";
    int Response;
    cin >> Response;
    return Response;
}
```

다음의 실례는 PromptAndGet()를 리용하여 두개의 입력값중에서 큰 값과 작은 값을 결정하는 프로그람토막이다.

```
int Value1 = PromptAndGet();
int Value2 = PromptAndGet();
cout << "Max: " << Max(Value1, Value2) << endl;
cout << "Min: " << Min(Value1, Value2) << endl;
```

5장에서는 문자가 모음인가를 결정하는 코드를 개발하였다.

쏘프트웨어의 재리용에 의하여 매 시각 계산하는데 필요한 명령문을 다시 쓰지 않고 함수에 코드토막을 추가하여 교갑화할수 있다. 이 함수의 이름은 IsVowel이다. 함수는 파라메터로 Char형값을 하나 가지며 그 문자가 모음인가를 가리키는 bool형값을 되돌린다.

```
//IsVowel():파라메터가 모음인가를 결정한다.
bool IsVowel (char ch) {
   switch (ch) {
```

```
        case 'a' : case 'A';
        case 'e' : case 'E';
        case 'i' : case  'I';
        case 'o' : case 'O';
        case 'u' : case 'U';
            return true;
        default:
            return false;
    }
}
```

다음의 실례에서는 어떤 수의 차례곱을 계산하는 함수를 정의한다. 이미 4장에서 론의한것처럼 차례곱식은 n!으로 표시한다.

$$n! = \begin{cases} 1 & n=0\text{인 경우} \\ n*(n-1)*\cdots*1 & n>1\text{인 경우} \end{cases}$$

이 차례곱을 계산하는 함수이름은 Factorial()이며 한개의 파라메터 n을 가지고 있다. 되돌림형은 int이다.

```
// Factorial(): 파라메터 n을 받아서 n차례곱을 계산한다.
int Factorial(int n) {
    int nfactorial = 1;
    while (n > 1) {
        nfactorial *= n;
        --n;
    }
    return nfactorial;
}
```

이 함수는 사용자로부터 n값을 받아서 차례곱을 계산한다. nfactorial원소에 처음부터 계산한 값이 루적되여 연산이 계속 진행된다. 한번 **while**순환이 진행될 때 n의 값을 하나씩 감소시킨다. n값이 1과 같거나 작다면 차례곱계산은 끝나게 된다. 결과값은 옹근수형으로 되돌린다. n이 값파라메터이기때문에 실제파라메터를 받는 함수로 변화시킬수 있다. 다음의 코드토막은 이러한 실례를 보여 주었다.

```
int i = 5;
int Result = Factorial(i);
cout << i << "! equals " << Result << endl;
```

출력결과는

```
5! equals 120
```

이다. Factorial()함수는 n값이 유효한것인가를 검사하지 않는다. 이러한 검사는 련습에서 취급한다.

6.4 2차다항식의 적분

n차다항식 f(x)는 a_ix^i , 0≦i≦n형식을 가진 n+1개의 항들로 구성된다. 여기서 a_n은 0이 아니다. n차다항식을 표시하는데는 아래와 같이 두가지 표준형식이 있다.

$$a_nx^n + a_{n-1}x^{n-1} + \cdots + a_2x^2 + a_1x + a_0$$

혹은

$$\sum_{i=0}^{n} a_ix^i$$

그림 6-3은 x^2 +2x+5의 그라프이다. 이 그라프에서 색칠된 부분은 x축의 1과 4구간의 면적이다. 미분방법을 리용하여 이 면적을 계산할수 있다.

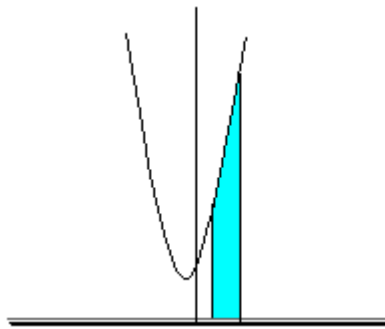



그림 6-3. x^2 +2x+5의 곡선에서 (1, 4)구간의 면적

∫기호는 곡면의 면적을 계산하는 적분기호이다. n차방정식에 대하여 x축구간 (s, t)의 곡선제형면적을 구하는 수학식은 아래와 같다.

$$\int_s^t a_nx^n + a_{n-1}x^{n-1} + \cdots + a_2x^2 + a_1x + a_0dx$$

총합은 다음과 같이 계산할수 있다.

$$\sum_{i=0}^{n} \frac{a_it^{i+1}}{i+1} - \sum_{i=0}^{n} \frac{a_is^{i+1}}{i+1}$$

이 공식을 다음과 같이 쓸수도 있다.

$$\sum_{i}^{n} \frac{a_i}{i+1}\left(t^{i+1} - s^{i+1}\right)$$

2차다항식에 대하여 이 공식은 다음과 같이 다시 쓸수 있다.

$$a_2\left(t^3 - s^3\right)/3 + a_1\left(t^2 - s^2\right)/2 + a_0(t - s)$$

프로그람 6-3에서 QuadraticArea()함수는 5개의 double형파라메터 a2, a1, a0, s, t(double형이다)를 가진 double형함수이다.

```
double QuadraticArea(double a2, double a1,
  double a0, double s, double t);
```

```
// 프로그램 6-3: X축의 곡면의 면적을 계산한다.
#include <iostream>
#include <string>
using namespace std;
double QuadraticArea(double a2, double a1, double a0, double s, double t);
int main() {
    cout << "Enter quadratic coefficients (a2, a1, a0): ";
    double a2;
    double a1;
    double a0;
    cin >> a2 >> a1 >> a0;
    cout << "Enter interval of interest(n1 n2): ";
    double t;
    double s;
    cin >> s >> t;
    double Area = QuadraticArea(a2, a1, a0, s, t);
    cout << "The area of quadratic polynomial"
      << a2 << "x * x + " << a1 << "x + " << a0
      << "for the x-interval (" << s << " " << t << " )"
      << "is" << QuadraticArea(a2, a1, a0, s, t) << endl;
}
double QuadraticArea(double a2, double a1, double a0, double s, double t) {
    double term2 = a2*(t*t*t - s*s*s)/3.0;
    double term1 = a1*(t*t - s*s)/2.0;
    double term0 =  a0*(t - s);
    return term2 + term1 + term0;
}
```

프로그람 6-3. 곡선제형면적계산

파라메터 a2, a1, a0은 2차방정식의 결수와 일치하며 s와 t는 x축구간을 나타낸다. 함수는 앞에서 본 공식을 리용하여 면적을 계산하고 그 총합을 되돌린다.

```
double term2 = a2 *(t*t*t - s*s*s) / 3.0;
double term1 = a1 (t*t - s*s) / 2.0;
double term 0 = a0 *(t - s);
return term2 + term1 + term0;
```

프로그람은 파라메터값들을 입력하여 자체로 곡선의 면적을 계산한다.

```
cout << "Enter quadratic coefficients (a2 a1 a0): ";
double a2;
double a1;
```

```
double a0;
cin >> a2 >> a1 >> a0;
cout << "Enter interval of interest(n1 n2): ";
double s;
double t;
cin >> s >> t;
```

QuadraticArea()함수를 호출하여 요구하는 유효범위의 면적을 정확히 계산한다.

```
double Area = QuadraticArea(a2, a1, a0, s, t);
```

련습에서 다른 차수를 가진 곡선의 제형면적을 구하는 문제를 취급한다.

6.5 국부유효범위

왜 프로그람 6-4가 콤파일되지 않는가 고찰해 보시오. 그것은 main()함수에서 a와 b를 참조할수 없기때문이다. 객체 a와 b는 main()함수의 유효범위에 존재하지 않는다. C++규칙에 따라 함수의 파라메터나 함수안에서 선언된 객체들은 함수 그자체안에서만 리용될수 있다. 이러한 객체들을 함수에 대하여 국부적이라고 한다. 어디서 정의되였으며 또 어떻게 정의하였는가에 따라 유효범위규칙은 또한 객체와 파라메터를 구분할수 있게 한다.

6.5.1 국부유효범위규칙

블로크는 대괄호안에 있는 명령문모임이다. 이 정의에 따르면 함수본체는 곧 블로크이다. C++에서는 명령문이 놓일수 있는 곳에 다른 명령문블로크를 배치할수 있다. 이러한 특성은 블로크안에 블로크를, 그안에 또 블로크를 포함할수 있게 한다. 다른 블로크안에 포함된 블로크를 겹싸인 블로크(nested block)라고 한다. 블로크를 포함하는 명령문모임은 오른쪽대괄호에 의해 자연적으로 경계가 그어 지며 C++용어로 말한다면 끝나게 된다. 따라서 오른쪽대괄호다음에는 반두점이 필요없다.

```
{
    int  a = 1;  //반두점이 필요하다
}   //반두점이 필요없다.
```

C++의 유효범위는 국부객체가 블로크 또는 겹싸인 블로크에서만 리용될수 있는가를 지적한다. 국부객체는 자기를 정의한 블로크내에서만 리용할수 있다.

```
// 프로그람 6-4. 유효범위를 설명
#include <iostream>
#include <string>
using namespace std;
void Mystery(int a, int b);
int main() {
    int  i = 10;
    int  j = 20;
```

```
    Mystery(i, j);
    cout << a << endl;
    cout << b << endl;
    return 0;
}
void Mystery(int a, int b) {
    cout << a << endl;
    cout << b << endl;
    a = 1;
    b = 2;
    cout << a << endl;
    cout << b << endl;
    return;
}
```

프로그람 6-4. 유효문제에 대한 프로그람

형식파라메터는 함수본체의 시작에 정의되였다. 이것은 형식파라메터가 함수본체에서 리용될수 있다는것을 의미한다.

6.5.2 객체이름의 재리용

유효범위규칙은 객체가 리용될수 있는 명령문들을 제한한다. 이 제한은 객체이름을 재리용하는것으로 해결하여야 한다. 물론 이름을 다시 재리용하라면 초학도들은 놀랄지 몰라도 이름을 재리용할수 없다면 프로그람작성이 얼마나 어려워 지겠는가를 상상해 보시오.

그러면 프로그람작성자들은 함수제작자와는 관계없이 쏘프트웨어과제에 있는 모든 함수들에서 리용되는 객체이름들을 다 알고 있어야 한다. 뿐만아니라 이러한 파제는 유연성을 보장하기 위하여 빈약한 이름들을 고쳐 주어야 하며 또 관리할수 없다는 편향이 생긴다.

C++는 서로 다른 블로크에 있는 객체들을 같은 이름으로 설정할수 있게 한다. 즉 다음의 코드가 그러한 실례로 된다.

```
int main() {
    int  a = b;
    int  b = 20;
    Mystery (a, b);
    cout << a << endl;
    cout << b << endl;
    return 0;
}
```

main()함수의 국부객체 a와 b는 앞서 서술한 Mystery()함수의 정의에서 형식파라메터 a, b와 차이난다. main()함수의 국부객체 a와 b는 오직 main()함수의 명령블로크에서만 존재하며 Mystery()의 형식파라메터 a, b는 Mystery()의 명령블로크안에서만 존재한다. 이러한 이름의 재리용은 두 함수의

활성화레코드에도 반영된다. 둘다 a와 b라는 원소를 가지고 있지만 서로 종속되여 있다.

main()	
a	10
b	20

Mystery()	
a	1
b	2

프로그람의 출력은 다음과 같다.

```
10
20
1
2
10
20
```

한 블로크가 다른 블로크를 포함하는 관계에 있는 두개의 블로크에 대해서도 이름의 재리용은 가능하다. 이름을 재리용하는 블로크가 끝나는 즉시 그를 둘러 싸고 있는 블로크에서 그 이름이 리용된다. 다음실례가 이 규칙을 보여 주고 있다.

```
{
    int i;
    // i를 리용하는 명령문은 int객체 i를 참조한다.
    {
        char i;
        // i를 리용하는 명령문은 char객체 i를 참조한다.
        {
            // i를 리용하는 명령문은 char객체 i를 참조한다.
            float i;
            // i를 리용하는 명령문은 float객체 i를 참조한다.
        }
        // i를 리용하는 명령문은 char의 char객체 i를 참조한다.
    }
    // i를 리용하는 명령문은 int객체 i를 참조한다.
}
// i가 여기서 우선적으로 사용될수 없다.
```

웃실례에서 이름의 중복은 여러개의 객체 i들이 서로 다른 자료형을 리용한다는 의미는 아니다. 실례로 다음의 코드는 이름도 형도 꼭 같은 객체를 정의하였다.

```
{ // 바깥블로크
```

```
    int i;
    // i를 리용하는 명령문은 바깔블로크의 i를 참조한다.
    { //내부블로크
        int i;
        // i를 리용하는 명령문은 내부블로크의 i를 참조한다.
    }
    //i를 리용하는 명령문은 바깔블로크의 i를 참조한다.
}
```

이름이 겹싸인 블로크에서 중복될수 있다고 하여 같은 블로크내에서 같은 이름으로 다시 정의할수는 없다. 즉 아래의 실례는 옳지 않다.

```
int i ;
float i;
```

또한 2개의 정의사이에 겹싸인 블로크가 끼여 있을 때에도 오유가 된다. 즉 다음의 실례는 오유이다.

```
int i;
{
    char i;
}
float i;
```

6.6 전역유효범위

객체는 명령문블로크안에서 정의할수 있지만 함수안에 또 다른 함수를 정의할수 없다. 그렇다고 함수원형선언의 리용에 모순되지 않는다. 원형은 함수의 대면부를 서술하는 선언이지 정의는 아니다. C++는 함수들이 전역유효범위에서 정의될것을 요구한다. 함수원형은 물론 객체정의와 선언, 형정의와 선언도 다 전역유효범위에서 할수 있다.

6.6.1 객체에 대한 유효범위규칙과 이름의 재리용

전역유효범위에서 정의된 객체를 전역객체라고 한다. 전역객체에 대한 유효범위규칙은 국부객체에서 비슷하다. 국부객체와 마찬가지로 전역객체는 그것이 정의된후 객체이름이 주어 진 블로크에서 중복되지 않는한 그 이름을 리용하여 참조될수 있다. 같은 블로크안에서 국부객체를 다시 정의할수 없는것처럼 전역유효범위의 한에서 전역객체는 중복될수 없다.

C++는 국부블로크안에서 이름이 중복되는 경우 전역객체를 리용할수 있도록 단항유효범위연산자 ::를 제공한다. 객체앞에 ::가 붙으면 리용되고 있는 객체가 전역객체라는것을 지적한다. 국부객체에 대하여 유효범위연산자를 리용할수 없다. 이 규칙을 리해하려면 다음의 실례를 보시오.

```
int i;
int main() {
    // i를 리용하는 명령문은 전역 int형의 객체 i를 참조한다.
    // ::i를 리용하는 명령문은 전역 int형의 객체 i를 참조한다. 이때
```

```
        //::이 필요없다.
        return 0;
    }
    void f() {
        // i를 리용하는 명령문은 int형의 전역객체 i를 참조한다.
        // ::i를 리용하는 명령문은 전역 int형의 객체 i를 참조한다. 이때
        // ::이 필요없다.
        char i;
        // i를 리용하는 명령문은 char형의 국부객체 i를 참조한다.
        // ::i를 리용하는 명령문은 int형의 전역객체 i를 참조한다.
        {
            // i를 리용하는 명령문은 char형의 국부객체 i를 참조한다.
            // ::i를 리용하는 명령문은 int형의 전역객체 i를 참조한다.
            int i;
            // i를 리용하는 명령문은 int형의 국부객체 i를 참조한다.
            // ::i를 리용하는 명령문은 int형의 전역객체 i를 참조한다.
        }
        // i를 리용하는 명령문은 char형의 국부객체 i를 참조한다.
        // ::i를 리용하는 명령문은 int형의 전역객체 i를 참조한다.
        return;
    }
    void g() {
        // i를 리용하는 명령문은 int형의 전역객체 i를 참조한다.
        // ::i를 리용하는 명령문은 int형의 전역객체 i를 참조한다.
        // 이때 ::이 필요없다.
        return;
    }
```

실례에서는 main()과 g()함수가 다 전역 int형객체 i를 리용할수 있다. 유효범위해결연산자 ::를 리용할수 있는데 반드시 필요한것은 아니다. 함수 f()에서 유효범위해결연산자를 리용하지 않아도 char형객체 i가 정의될 때까지 전역객체 i를 리용할수 있다. 이때부터는 유효범위해결연산자를 리용하여야만 전역객체 i를 리용할수 있다.

함수 f()안에 있는 블로크들에서는 국부객체 i가 리용상태로 있다. 이 객체는 전역객체 i와 구별된다. 국부객체의 변화는 전역객체에 영향을 주지 않는다. 유효범위해결연산자는 또한 함수들과 사용자정의형들에서도 제공될수 있다. 이것은 보통 클라스정의에서만 쓸모 있다. 마지막장에서 유효범위해결연산자의 리용방법을 보여 준다.

6.6.2 객체의 초기화

국부객체와는 달리 전역객체는 언제나 초기화된다. 기본자료형전역객체가 초기화가 명백히 되지 않으면 값 0으로 초기화된다. 실례로 다음의 프로그람은 0을 현시한다.

```
#include <iostream>
#include <string>
using Namespace std;
int a;
int main() {
    cout << a << endl;
    return 0;
}
```

문 제

1. 아래의 명령문을 리용하는 함수의 형은 무엇인가?

```
return ;
```

2. 값파라메터를 const로 선언할 필요가 왜 없는가?
3. 두개의 론리형인수 x와 y를 가지고 x 또는 y를 되돌리는 bool형함수 nor()를 작성하시오.
4. 두개의 옹근수를 가지고 평균값을 계산하는 함수를 작성하시오.
5. 옹근수값을 돌리는 함수 compare()를 작성하시오. 함수는 두개의 옹근수형 인수 a와 b를 비교하여 a<b이면 -1, a>b이면 1, a=b이면 0을 돌린다.
6. 다음 프로그람의 결과는 무엇인가.

```
#include <iostream>
using Namespace std;
int main() {
    int i = 10;
    int j = 12;
    cout << i << "  " << j << endl;
    {
        i++;
        int i = 3;
        cout << i << "  " << j << endl;
        {
            int j = 11;
            i++;
            cout << i << "  " << j << endl;
        }
    j++;
    cout << i << "  " << j << endl;
    return 0;
    }
}
```

7. 다음의 프로그람의 출력결과는 무엇인가?

```
#include <iostream>
using Namespace std;
int a = 0;
void Demo() {
    int b = 4;
    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
}
int main() {
    int a = 2;
    int b = 3;
    cout << "b = " << b << endl;
    cout << "a = " << a << endl;
    Demo();
    cout << "b = " << b << endl;
    return 0;
}
```

8. 다음의 프로그람의 출력결과는 무엇인가?

```
#include <iostream>
using Namespace std;
int a = 0;
void Demo() {
    int b = 4;
    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
}
int main() {
    int a = 2;
    int b = 3;
    cout << " b = " << b << endl;
    cout << " a = " << ::a << endl;
    Demo();
    cout << "b = " << b << endl;
    return 0;
}
```

9. 반복처리에 의하여 많은 문제들이 콤퓨터상에서 쉽게 해결될수 있다. 사용자는 초기값을 주고 프로그람은 요구하는 정확도로 대답을 준다. 복잡한 문제에 대하여 반복처리는 일반적이며 여러 갈래의 어

려운 문제를 해결하는 강력한 방법이다. 관을 통하는 물흐름의 깊이를 결정하는 기술문제를 해결하여 보자. 관을 통하여 흐르는 물흐름은 다음의 공식으로 서술된다.

$$Q = \frac{1.49}{N} AR^{2/3} S^{1/2}$$

통로는 수직통로이며 벽의 너비는 15피트이다. 깊이는 10피트이며 따라서 경사도는 0.0015, 그리고 쓸림결수는 0.014이다. 이때 물이 초당 1000피트립방으로 흐를 때 물의 깊이는 얼마인가? 이 문제를 해결하려면 사용자가 깊이를 추측하고 적당한 흐름을 계산하도록 하는 프로그람을 설계하고 실행하여야 한다. 흐름이 지내 작으면 사용자는 약간 높여 깊이 측정하며 너무 높으면 조금 낮추어 측정한다. 측정은 계산된 흐름이 0.1%오차를 가진 요구되는 흐름이 될때까지 반복된다.

$$R = (depth \times Width) \div (2.0 \times depth + Width)$$

관은 수직벽을 가지고 있는데 그 너비는 15피트이다. 관의 깊이가 10피트이면 0.0015피트의 경사도를 가지며 쓸림결수는 0.014이다. 초당 백립방피트의 물이 관을 지나갈 때 물은 어느 정도로 깊어 지는가? 이 문제를 풀기 위해서 사용자는 물의 깊이를 알고 그다음 그 흐름량을 계산하는 프로그람을 설계하고 작성하여야 한다. 흐름량이 너무 작다면 사용자는 깊이가 덜 깊다는것을 알고 있어야 한다. 만일 흐름량이 높다면 사용자는 얕은 깊이라는것을 알고 있어야 한다.

주의

전역객체의 제한성

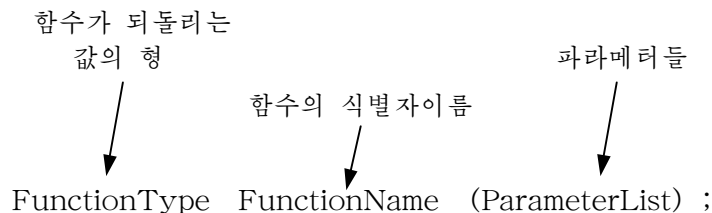
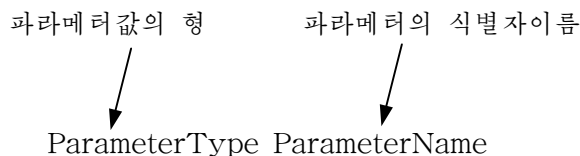
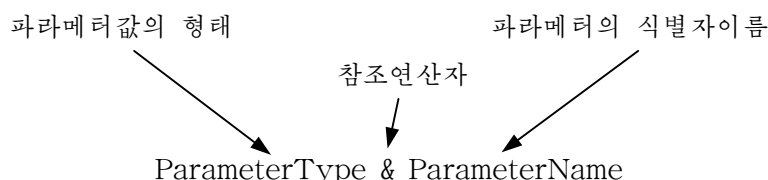
전역객체가 어디서 변하는지 프로그람작성자가 추적할수 없는 경우가 있다. 다음의 경우를 생각해 보시오.
함수 f()는 그 본체안에서 전역객체를 바꾸지도 리용하지도 않았다. 그러나 f()함수가 호출한 함수에서는 전역객체가 변하였다. 이 과정을 추적한다는것은 매우 어렵다. 이런 어려움을 피하기 위하여 프로그람작성방법론에 따라 전역객체의 리용을 제한한다. 6.15에 있는 증권도표문제의 실현에서 전역유효범위에 선언된 SimpleWindow객체를 요구한다는것을 알아 두시오.

6.7 참조파라메터

값파라메터에 의한 파라메터넘기기의 우점의 하나는 프로그람의 리해를 간단하게 한다는것이다. 파라메터를 값으로 넘기면 호출된 함수는 실제파라메터의 값을 바꿀수 없다. 실례로 값파라메터를 가진 함수호출을 만날 때 그 함수의 본체가 어떻게 되여 있든지 관계없이 실제파라메터는 호출전이나 함수가 귀환된후에나 다 같은 값을 가진다. 다음의 코드토막을 보시오.

```
//형부분
float Flow(float Depth, float Width);
...
int main() {
    ...
    f = Flow(CurrentDepth, ChannelWidth);
    ...
}
```

우의 실례에서는 Flow함수본체를 시험해 보지 않고도 원형선언만 보면 CurrentDepth와 CurrentWidth가 변하지 않는다는것을 알수 있다. 파라메터를 수정하지 않고도 되돌림값으로 함수를 호

출한 명령문에 있는 값을 변화시키는 함수를 부작용이 없다고 말한다. 부작용이 없는 함수를 리용하는 프로그람이 부작용을 가진 프로그람보다 리해하기가 쉬우므로 가능한껏 값의 호출을 리용하게 된다. 그러나 때때로 함수에 넘겨 지는 실제파라메터를 변화시킬 필요가 있을 때도 있다. 실례로 함수를 설계할 때 하나이상의 값을 귀환시킬 경우가 있다. 이를 위하여 C++에서는 참조파라메터를 제공한다. 값파라메터와는 달리 참조파라메터는 실제파라메터의 참조 즉 지적자를 넘긴다. 따라서 형식참조파라메터가 함수안에서 변화될 때 실제파라메터도 역시 변한다. 값파라메터를 가진 함수대면부의 형식을 다시 보자.

여기서 ParameterList는 다음와 같은 형식으로 선언한다.

파라메터가 참조파라메터라는것을 가리키기 위해 다음과 같은 확장된 파라메터 선언문법을 보여 준다.

여기서 &기호는 파라메터값의 복사가 아니라 그의 참조를 가리킨다.

참조형파라메터의 리해를 위하여 MyStery프로그람을 다시 보기로 하자. 초기프로그람에서는 Mystery의 대면부가

```
void Mystery( int a, int b);
```

였다. 즉 Mystery()는 2개의 파라메터를 값으로 넘긴다. 더우기 Mystery()는 되돌림값이 없다. 즉 **void**함수이다. 참조형파라메터를 리용하기 위하여 Mystery()의 대면부를

```
void Mystery(int a, int &b);
```

로 바꾸겠다. 첫 파라메터 a는 여전히 값파라메터이지만 b는 참조형파라메터이다. 프로그람 6-5는 Mystery()를 리용하는 프로그람이다. 이 프로그람은 값파라메터와 참조파라메터사이의 차이를 보여 준 것이다. 프로그람 6-5의 결과는 다음과 같다.

```
Output at beginning of main
i is 10
```

```
j is 20
Output at beginning of Mystery
a is 10
b is 20
Output after Mystery assignments
a is 5
b is 6
Output after Mystery returns
i is 10
j is 6
```

따라서 Mystery()가 b를 변화시키면 main()의 j가 변화된다.

```
#include <iostream>
#include <string>
using Namespace std;
void Mystery(int a, int &b) {
    cout << "Output at beginning of Mystery" << endl;
    cout << "a is " << a << endl;
    cout << "b is " << b << endl;
    a = 5;
    b = 6;
    cout << "Output after Mystery assignments"<< endl;
    cout << "a is "<< a << endl;
    cout << "b is "<< b << endl;
    return;
}
int main() {
    int i = 10;
    int j = 20;
    cout << " Output at beginning of main"<< endl;
    cout << "i is "<< i << endl;
    cout << "j is "<< j << endl;
    Mystery(i, j);
    cout << " Output after Mystery returns"<< endl;
    cout << "i is "<< i << endl;
    cout << "j is "<< j << endl;
    return 0;
}
```

프로그램 6-5. 참조에 의한 호출실례프로그람

정확한 리해를 위하여 main()과 Mystery()함수의 활성화레코드를 보기로 하자. 그림 6-4의 왼쪽그림은 Mystery()에서 값주기명령문이 실행되기전의 활성화레코드를 보여 준다. 그림에서 화살표는 Mystery()가 b를 호출할 때 실지로 main()함수에 있는 j를 호출한다는것을 가리킨다. 즉 Mystery()에 있는 b를 참조할 때 실지로 main()에 있는 j를 참조한다고 말할수 있다.

참조형파라메터를 통하여 넘겨 지는 객체들은 참조로서 넘어 간다.

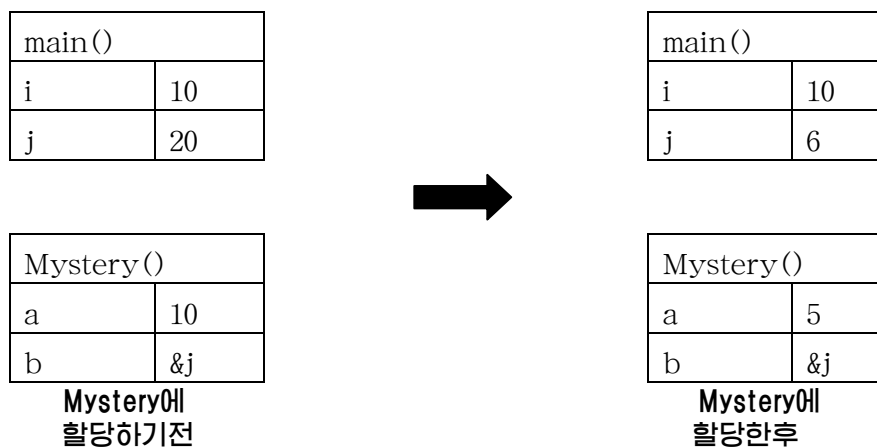



그림 6-4. Mystery()와 main()의 활성화레코드

그림 6-4의 오른쪽은 Mystery()에서 값주기명령문을 실행한후의 활성화레코드이다. 그림에서 보면 main()함수의 j가 변하였다는것을 알수 있다. 한편 i는 변하지 않았다. 이러한 결과는 a가 값파라메터이기때문에 참조형식이 아니라 복사형식으로 넘어 간 결과이다.

참조형파라메터의 실용성에 대해 리해하기 위하여 4장에 있는 프로그람을 보자. 이 프로그람은 3개의 값을 받아 분류한 값을 출력한다. 적당한 순서규칙을 얻기 위하여 3개의 값의 가능한 6개의 규칙을 모두 검사하며 적합한 값주기명령문들을 리용하여 값들에 대한 정확한 순서를 얻는다. 이때 참조형파라메터를 가진 함수를 리용하여 훨씬 적은 코드를 가지고 같은 효과를 얻을수 있다. 목적은 3개의 옹근수값을 작성하는데 있다. 함수는 3개의 옹근수참조파라메터값들을 접수하여 값들을 묶음하고 함수가 귀환할 때 첫 파라메터에는 제일 작은 값이, 두번째 파라메터에는 그 다음값이, 세번째 파라메터에는 제일 큰 값이 들어 가도록 한다.

여기서 리용한 방법은 값들을 두개씩 비교하고 순서에 맞게 교체하는것이다. 먼저 첫번째와 두번째 파라메터를 비교하고 순서에 맞게 되여 있지 않으면 값들을 교체한다. 이 조작은 첫번째와 세번째 파라메터에 대해서도 반복한다. 이 두 단계를 거치면 첫 파라메터에는 가장 작은 값이 들어 가게 된다. 마지막단계에서 두번째와 세번째 파라메터를 비교하고 순서가 맞지 않으면 교체한다. 값이 만일 같은 경우에는 값을 교체하지 않는다.

이와 같은 과정을 거쳐 3개의 값을 순서대로 묶음한 결과 여기에는 3번의 비교와 비교결과에 따르는 3번의 교환조작이 포함된다. 다음의 코드가 이와 같은 동작을 수행한다.

```
void Sort3(int &a, int &b, int &c) {
    if(a > b) {
        int t = a;
        a = b;
        b = t;
```

```
        }
        if(a > c) {
            int t = a;
            a = c;
            c = t;
        }
        if(b > c) {
            int t = b;
            b = c;
            c = t;
        }
        return;
    }
```

함수가 정확히 동작하려면 3개의 파라메터가 모두 참조파라메터여야 한다. 그렇게 하지 않으면 함수에서 파라메터값을 아무리 바꾸어도 실제파라메터는 변하지 않는다. 우의 함수를 리용하여 프로그람 4-3를 고쳐 프로그람 6-6으로 만들면 초기의 프로그람보다는 리해하기가 더 쉽고 코드작성도 적어 진다.

```
#include <iostream>
#include <string>
using namespace std;
void Sort3(int &a, int &b, int &c) {
    if(a > b) {
        int tmp = a;
        a = b;
        b = tmp;
    }
    if(a > c) {
        int tmp = a;
        a = c;
        c = tmp;
    }
    if(b > c) {
        int tmp = b;
        b = c;
        c = tmp;
    }
    return;
}
int main() {
```

```
    cout << "Please provide three integers:"
    int Input1;
    int Input2;
    int Input3;
    cin >> Input1 >> Input2 >> Input3;
    int Output1 = Input1;
    int Output2 = Input2;
    int Output3 = Input3;
    Sort3(Output1, Output2, Output3);
    cout << Input1 << " " << Input2 << " "<< Input3
      << "in sorted order is "
      << Output1 << " "  << Output2 << " " << Output3 << endl;
    return 0;
}
```

프로그램 6-6. 3개의 수를 분류하는 프로그램

프로그램의 조작과정을 확고히 리해하려면 main()함수와 Sort3()의 활성화레코드를 보시오. 프로그람을 실행시켜 20, 5, 9라는 값을 입력하였을 때 Sort3()함수의 활성화레코드는 그림 6-5의 왼쪽그림과 같다.

Sort3()의 파라메터들인 a, b, c는 Main()함수의 Output1, Output2, Output3의 참조이다. Sort3()은 a와 b의 값을 비교하고 a가 b보다 크기때문에 그것들을 교체한다. 실제는 Output1과 Output2의 값이 교체된다. 이 단계를 지난 다음의 활성화레코드는 그림 6-5의 가운데그림이다. 그림을 보면 Output1과 Output2의 값이 교체되였다는것을 알수 있다.

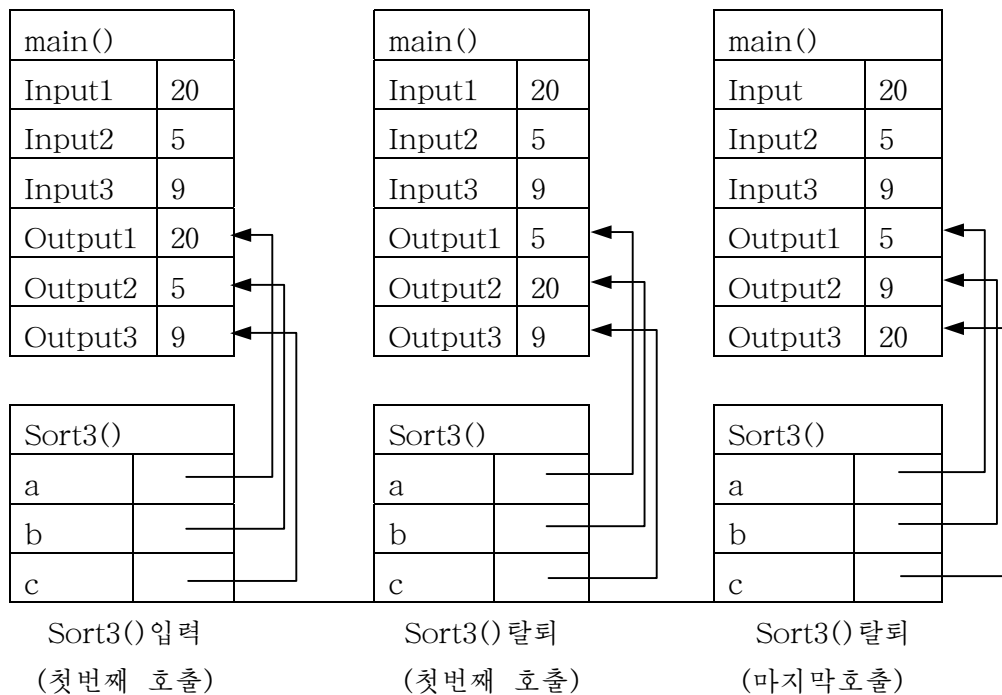



그림 6-5. 프로그람 6-6의 실행시 활성화레코드

다음 a와 c의 값이 비교된다. 이것들의 순서가 맞으므로 아무러한 동작도 진행되지 않는다. 그다음 b와 c가 비교된다. 순서가 틀리므로 교체된다. Sort3()이 main()으로 돌아 가기 직전에 활성화레코드는 그림 6-5의 오른쪽그림과 같다. 즉 모든 수들이 순서대로 묶음되였다. 즉 프로그람의 출력결과는 다음과 같다.

```
20 5  9 in sorted order is 5 9 20
```

프로그람은 더 간단히 할수 있다. Sort3()은 교체라고 하는 같은 조작을 서로 다른 3개의 조의 수들에 대하여 진행한다. 만일 이 기능을 수행하는 함수를 작성할수 있다면 교환부분을 모두 교환함수로 교체할수 있다. 이 함수를 리용하면 코드작성이 작고 단순해 진다. 두개의 값을 교환하는 함수에는 참조파라메터를 리용하는것이 쉽다. 그 함수를 아래에 주었다.

```
void Swap(int &x, int &y) {
    int tmp = x;
    x = y;
    y = tmp;
    return;
}
```

프로그람 6-7은 완성된 분류프로그람코드이다. 이 프로그람은 참조파라메터를 가진 보조프로그람을 호출하며 또한 그 보조프로그람이 다른 보조프로그람을 호출하는것으로 하여 아주 흥미 있는 프로그람이다. 활성화레코드가 무엇처럼 보이는가?

앞실례의 입력경우를 가지고 전에 하던것처럼 프로그람을 한 걸음씩 분석할수 있다. Sort3()은 파라메터 a와 b를 비교하고 순서가 맞지 않다는것을 결정한다. 이 프로그람에서는 그 파라메터의 값을 직접 교체하는것이 아니라 Swap()를 호출하여 교환한다. Swap()의 형식파라메터 x와 y는 다 참조파라메터이다. 즉 a와 b에 대한 참조이다. 그러나 a와 b구조체도 참조파라메터이므로 이것들은 그 어떤 객체에 대한 참조이다.

```
#include <iostream>
#include <string>
using Namespace std;
void Swap(int &x, int &y) {
    int tmp = x;
    x = y;
    y = tmp;
    return;
}
void Sort3(int &a, int &b, int &c) {
    if(a > b)
        Swap(a, b);
    if(a > c)
        Swap(a, c);
```

```
    if(b > c)
        Swap(b, c);
    return;
}
int main() {
    cout << "Please enter three integers: " ;
    int Input1;
    int Input2;
    int Input3;
    cin >> Input1 >> Input2 >> Input3;
    int Output1 = Input1;
    int Output2 = Input2;
    int Output3 = Input3;
    Sort3(Output1, Output2, Output3);
    cout << Input1 << " "<< Input2 << " " << Input3
      << "in sorted order is "
      << Output1 << " " << Output2 << " " << Output3 << endl;
    return 0;
}
```

프로그람 6-7. Swap()함수에 의한 3개수의 분류

그림 6-6에 있는 왼쪽그림은 Swap()의 첫번째 명령문이 실행되기 직전의 활성화레코드이다. 파라메터 x, y는 main()의 국부객체 Output1과 Output2에 대한 참조이다. Swap()의 기능은 그 값들을 교환하는것이다.

그림의 가운데그림은 Swap()가 완료한후 활성화레코드인데 Sort3()으로 돌아 가기 직전의 상태이다. Swap()함수는 다시 호출되여 b와 c의 값을 교환한다. 그림 6-6의 오른쪽그림은 Swap()가 이 작업이 끝난후의 활성화레코드이다. 이 그림에서 x와 y는 Output2와 Output3에 귀착된다. 그림들을 보면 어느것이 값파라메터이고 어느것이 참조파라메터인가를 명백히 알수 있지만 매번 화살표를 그린다는것은 어색한 일이다.

참조파라메터에 대하여 생각하는 단순한 방법은 형식파라메터의 이름이 실제파라메터의 별명이라는것이다. Sort3()에서 a는 main()에 있는 Output1의 별명이다. 즉 a의 값을 얻거나 변경할 때 Output1의 값도 얻거나 변경된다. 이것을 도입한다면 형식참조파라메터를 실제파라메터에 련결하는 선을 그리지 않고 형식파라메터의 값에 일치하는 칸에 실제파라메터의 이름을 배치하여 그림을 간단히 할수 있다. 그러나 값파라메터의 값이 아니라 객체에 대한 참조라는것을 보여 주는 방법이 필요하다.

이름앞에 &기호를 붙여 실제파라메터가 객체에 대한 참조라는것을 지적할수 있다. 그림 6-7에 이 표기법을 주었다. 이 그림은 Sort3()함수에서 a를 호출할 때 실지로는 Main()함수의 Output1을 참조하며 마찬가지로 Swap()에서 형식파라메터 x는 Sort3()의 Output2를 참조한다는것을 보여 준다.

그림 6-6. Swap()를 리용한 분류프로그람의 활성화레코드

main()	
Input1	20
Input2	5
Input3	9
Output1	5
Output2	9
Output3	20

Sort3()	
a	&Output1
b	&Output2
c	&Output3

Swap()	
X	&Output2
Y	&Output3
tmp	20

Swap() 탈퇴
(마지막호출)

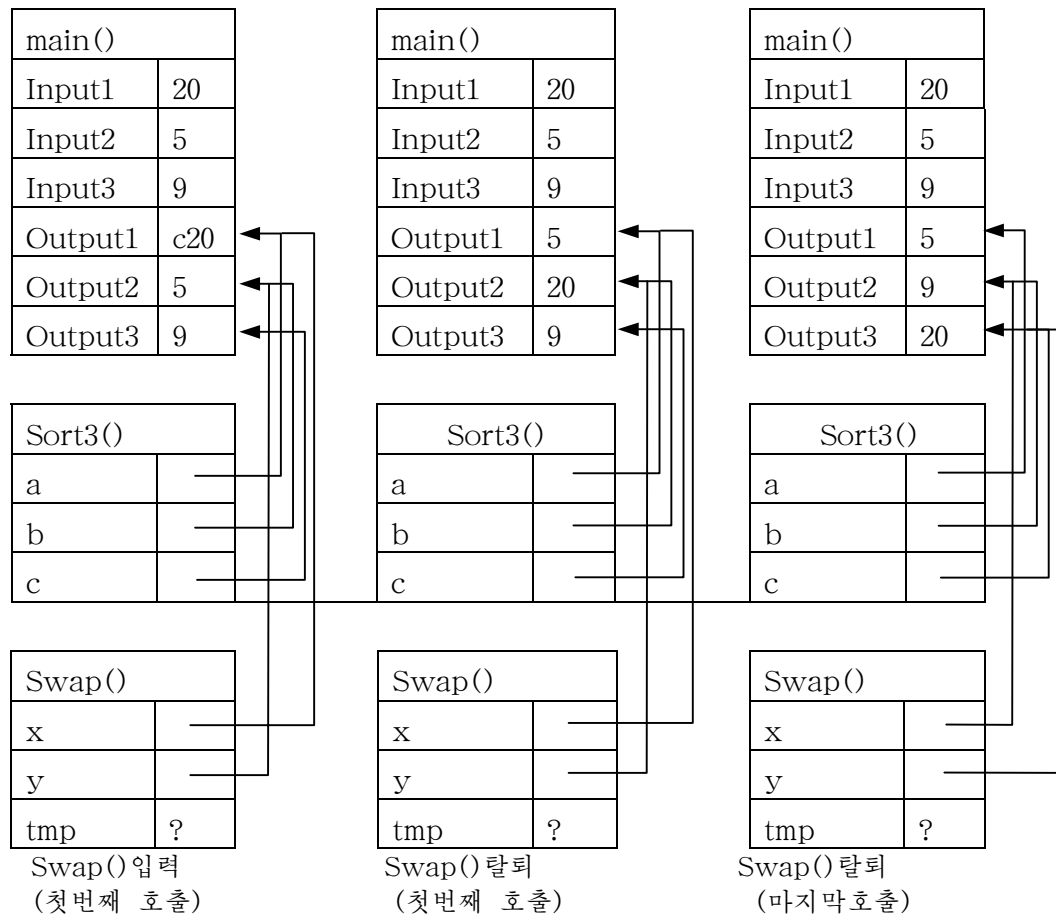
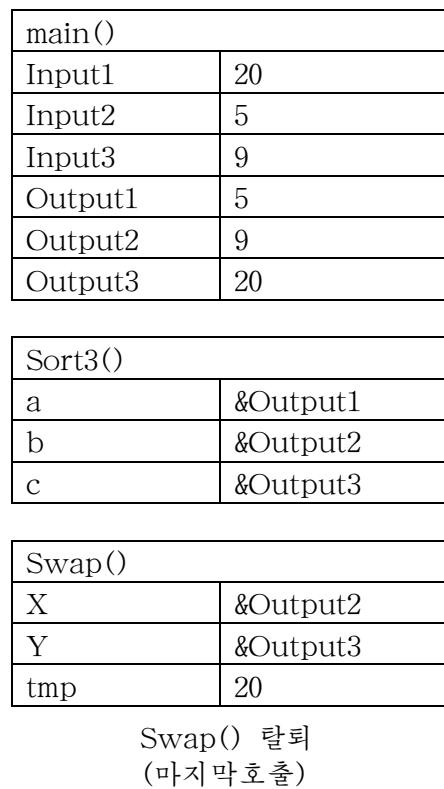



그림 6-7. Swap()를 리용한 분류프로그람의 활성화레코드

6.8 참조에 의한 객체의 넘기기

참조파라메터를 리용하는것이 반드시 필요한 경우가 바로 함수에 흐름객체를 넘길 때이다. 물론 그 리유는 흐름에 자료를 입력하고 출력할 때 흐름을 변경하기때문이다. 이 변화가 흐름객체에 반영되여야 한다.

알고 있는것처럼 흐름객체를 값으로 넘긴다면 흐름객체에 대한 그 어떤 변경도 흐름객체의 복사본에 대하여 이루어 지지만 실제객체에 대해서는 이루어 지지 못한다. 따라서 함수에 흐름을 넘길 필요가 있을 때마다 흐름을 참조로 넘겨야 한다.

사실 거의 모든 콤파일러들은 흐름객체가 함수에 넘어 갈 때 흐름이 참조로 넘었는가를 확인한다. 그렇지 않은 경우는 콤파일러오유가 생기고 객체파일은 만들어 지지 않는다.

다음의 함수는 흐름객체의 참조실례이다.

```
void OutputValue(ostream &out, int Value) {
    cout << "Value is" << Value << end1;
    return;
}
```

다음의 코드는 OutputValue()함수를 리용하여 흐름 cout로 k라는 값을 출력한다.

```
OutputValue(cout , k);
```

마찬가지로

```
OutputValue(cerr, k);
```

는 cerr흐름에 k값을 출력한다.

흐름에 값을 출력하는 함수는 프로그람의 여러 위치에서 표식정보와 함께 여러개의 값을 출력할 필요가 있을 때 쓸모 있다. 이런 함수를 리용하여 자료를 표시기에 출력하는것은 물론 log파일에도 쓸수 있다. 또한 프로그람의 오유수정에도 쓸수 있다. 프로그람에서 필요한 위치에 있는 중요한 값들을 출력하는데 함수를 리용할수 있다. 여기서 시험해 보자.

실례로 프로그람 6-7에 있는 Sort3()을 《감시》하려고 한다고 가정하자. 그러면 함수의 매 단계마다 a, b, c의 값을 보아야 한다. 매 **if**명령문다음에 3개의 출력명령문을 출력하지 않고 3개의 값을 출력하는 함수로 호출할수 있다. 함수는 될수록 일반적이여야 하므로 파라메터들중 하나는 출력을 담당한 흐름으로 한다. 함수이름은 ShowValues()이다.

```
void ShowValues(ostream &out , int a, int b, int c) {
    Out << "a: " << a << endl;
    Out << "b: " << b << endl;
    Out << "c: " << c << endl;
    return ;
}
```

이 함수를 리용하여 clog흐름에 값들을 쓰기하는 Sort3()함수의 실례를 아래에 주었다.

```
void Sort3(int &a, int &b, int &c) {
    if(a > b)
```

```
        Swap(a, b);
    ShowValues(clog, a, b, c);
    if(a > c)
        Swap(a, c);
    ShowValues(clog, a, b, c);
    if(b > c)
        Swap(b, c);
    ShowValues(clog, a, b, c);
    return;
}
```

함수에 파라메터로서 입력흐름을 넘기는 능력을 사용할수 있다. 실례로 파일에 있는 자료를 처리하는 프로그람을 작성한다고 가정해 보자. 프로그람을 건반에서 자료를 입력하도록 개발한다면 검사를 보다 빨리 진행한다. 서로 다른 자료에 대하여 프로그람을 검사할 때마다 검사파일을 작성하는것보다 자료를 건반으로 입력한다면 프로그람검사가 빨라 질수 있다. 물론 프로그람검사가 끝나면 파일에서 자료를 읽어 들이도록 하여야 한다. 이제 작성하게 될 프로그람은 각이한 흐름에 대하여 같은 조작을 하여야 한다. 이것은 자료를 입력한 흐름을 파라메터로 하는 함수를 리용하여 실현할수 있다.

이렇게 하면 각이한 원천으로부터 자료를 얻으려고 할 때 함수를 변경시킬 필요가 없어 진다. 아래에 함수 ReadValue()를 정의하였다.

```
bool ReadValues(istream &in, int &v1, int &v2, int &v3) {
    if (in == cin)
    cout << "Please enter three numbers";
    if (in >> v1 >> v2 >> v3)
        return true;
    else
        return false;
}
```

함수는 거의나 단순하다. 그러나 두개의 **if**명령문은 론의되여야 한다. ReadValue()함수가 입력에 리용된다면(실례로 검사 등을 들수 있다.) 자료의 입력재촉문을 출력하여야 한다. 반대로 파일에서 자료를 읽어 들인다면 이러한 재촉문이 필요없다. 실례로 **if**명령문이 바로 이 요구를 처리한다. 즉 자료입력과 진행되는 흐름이 cin(즉 건반)이라면 재촉문이 출력된다. ReadValue()함수가 보다 큰 체계로서 리용되여야 하므로 호출하는 함수에 3개 자료를 성과적으로 읽어 들일수 있는가를 알려 주어야 한다.

5장에서 본것처럼 입력식은 입력조작이 성공이면 true, 아니면 false를 되돌린다. 두번째 **if**명령문은 3개 값이 다 읽어 졌는가를 검사한다. 3개 값을 성과적으로 읽어 들이면 true아니면 false를 되돌린다. ReadValue()를 리용하면 프로그람 6-7을 test.dat라는 파일에서 값을 읽어 들이도록 수정할수 있다.

프로그람 6-8은 교정된 함수를 리용한다. 함수는 파일이 열렸는가, 그리고 ReadValue()함수가 true인가를 검사한다. 이것들중 어느 하나라도 실패하면 프로그람은 cerr에 오유통보를 내고 조작체계에 하나를 되돌려서 오유가 발생했음을 알려 준다. 이런 형의 《방어프로그람작성법》은 다른 프로그람작성자들이 리용할 코드를 작성할 때 필요한 개념이다.

6.9 전화호출부호의 확인

많은 조작들이 자기 전화체계에 망라된 사용자들에게 먼저 유효한 호출을 제공하도록 요구한다. 이러한 안전성측정은 먼 거리호출을 보호하도록 한다. 내부전화체계와 외부를 련결하는 콤퓨터는 호출코드를 검사한다. 간단한 호출과 검사방법의 실례가 사용자가 5자리수를 입력하는것이다.

첫 세자리가 함께 추가된다. 이 합을 네번째 수자로 나누었을 때 나머지가 다섯번째 수자와 같아야 한다.

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main() {
    ifstream fin("test.dat");
    if (!fin) {
        cerr  << "Could not open test.dat:"<< endl;
        return 1;
    }
    int Input1;
    int Input2;
    int Input3;
    if(!ReadValues(fin, Input1, Input2, Input3)) {
        cerr  <<"Could not  read three Values"<< endl;
        return 1;
    }
    int Output1 = Input1;
    int Output2 = Input2;
    int Output3 = Input3;
    Sort3(Output1, Output2, Output3);
    cout << Input1 << " " << Input2 << " " << Input3
      << "in sorted order is "
      << Output1 << " " << Output2 << " " << Output3 << endl;
    return 0;
}
```

프로그람 6-8. 파일로부터 자료를 읽어 분류하는 프로그람

이 경우 사용자는 전화호출을 할수 있다. 그렇지 않으면 거절당한다. 수렬이 유효한가 아닌가를 결정하는 알고리듬이 비밀로 있는한 우연적으로 입력한 수가 들어 맞을 확률은 작아 진다.

프로그람 6-9는 이것을 실현한 례이다. iostream과 ctype서고를 참조하기 위한 include지령이 있고 그다음 bool형함수인 Get()와 Valid()를 원형선언하였다.

Get()함수의 목적은 5개의 참조파라메터를 사용자가 준 5개의 수자로 설정하는것이다. Valid()함수의 목적은 입력된 수자를 검사하고 그것이 유효한가를 검사한다.

main()함수는 입력자료를 보관할 5개의 int객체정의로부터 시작한다. Get()함수를 호출하고 그것이 성공이면 Valid()를 호출한다. main()함수로 이 함수의 되돌림값을 리용하여 코드가 유효한가를 식별한다. 이미 말한바와 같이 프로그람이 0을 돌리면 성공이고 아니면 실패이다. 0이 아닌 값들은 프로그람이 어떻게 실패하였는가를 지정한다. main()도 이러한 규칙을 따른다. Get()와 Valid()가 다 true를 돌리면 0을 돌리고 아니면 1을 돌린다.

```
// 프로그람 6-9: 전화호출코드의 유효성검사
#include <iostream>
#include <string>
#include <ctype.h>
bool Get(istream &in, int &d1, int &d2, int &d3, int &d4, int &d5);
bool Valid(int d1, int d2, int d3, int d4, int d5);
int main() {
    int d1;
    int d2;
    int d3;
    int d4;
    int d5;
    if(Get(cin, d1, d2, d3, d4, d5) && Valid(d1, d2, d3, d4, d5))
        return 0;
    else
        return 1;
}
bool Get(istream &sin, int &d1, int &d2, int &d3, int &d4, int &d5) {
    char c1;
    char c2;
    char c3;
    char c4;
    char c5;
    if(sin >> c1>> c2 >> c3 >> c4 >> c5) {
        if(isdigit(c1) && isdigit(c2) && isdigit(c3) && isdigit(c4)
          && isdigit(c5)) {
            d1 = c1 - '0'
            d2 = c2 - '0'
            d3 = c3 - '0'
            d4 = c4 - '0'
            d5 = c5 - '0'
            return true;
        }
        else
            return false;
    }
    else
        return false;
}
bool Valid(int d1, int d2, int d3, int d4, int d5) {
```

```
    if(d4 == 0)
        return false;
    else
        return ((d1 + d2 + d3) % d4) == d5;
}
```

프로그람 6-9. 전화호출코드의 유효성검사프로그람

Get()함수에서 입력자료는 cin흐름에 입력되는데 이 호출은 첫 파라메터로 넘어 간다. 사용자가 맞지 않는 값을 입력하는 경우에 char형객체로서 자료가 입력된다. 함수는 입력조작이 입력이 성공일 때만 0아닌 값을 되돌린다는 사실을 리용한다. 따라서 사용자가 5개의 값을 리용하지 않았다면 실행은 else명령문에로 돌아 가며 함수는 false를 되돌려 입력이 실패했다는것을 지적한다. 사용자가 5개의 입력값을 다 주었을 때 Get()함수는 그것들을 서고함수 isdigit()를 호출하여 수자들로 바꾼다. 이 함수는 그 파라메터가 10진수이면 true를 돌리고 아니면 false를 돌린다.

입력한 문자들이 모두 수자들이면 참조파라메터들은 적당한 수자들로 설정된다. 이 경우 함수가 성공이므로 true를 돌린다. 입력한 값이 이 수자가 아니면 함수는 else명령문을 수행하여 false를 돌린다.

Get()를 리용하여 수자모임들이 ASCII형식이라고 가정하면 2장에서 취급한것처럼 ASCII수자, 문자는 련속 커지는 순서로 묶어 진다(즉 '1'= '0'+1, '2'= '1'+1 등). Get()함수는 수자문자에서 0문자를 덜어 수값을 계산한다. 실례로 '3'-'0'은 옹근수 3이다.

프로그람 6-9는 그다음 Valid()함수를 정의하는데 이 함수의 기능은 값파라메터들을 검사하고 유효한 호출코드이면 true를, 그렇지 않으면 false를 돌린다. 유효성검사처리의 첫 단계는 네번째 수자를 검사하는것이다. 즉 네번째 수자는 0이 되면 안된다. 왜냐하면 호출알고리듬을 따른다면 네번째 수자는 나눔수로 리용되므로 0이 되여서는 안된다. 만일 4번째 수자가 0이면 Valid()는 false를 돌린다. 0이 아니라면 첫 세 수자를 합한다. 이 합을 네번째 수자로 나눈 나머지가 5번째 수자와 같다면 valid는 true를 돌린다. 아니면 false를 돌린다.

문 제

10. 다음의 프로그람을 보고 함수 f()의 원형을 선언하시오. 프로그람의 실행결과는 x is 2이다.

```
#include <iostream>
using namespace std;
int main() {
    int x = 1;
    f(x);
    cout << "x is " << x << endl;
    return 0;
}
```

11. 다음의 프로그람의 결과는 무엇인가?

```
#include <iostream>
using namespace std;
int f(int a, int &b) {
    int t;
```

```
    t = b;
    a = b;
    b = a;
}
int main() {
    int x = 10, y = 20;
    f(x, y);
    cout << "x is " << x << endl;
    cout << "y is " << y << endl;
    return 0;
}
```

12. 다음의 프로그램의 결과는 무엇인가?

```
void x(int a, int &b, int &c) {
    a = c;
    b = a;
    c = b;
}
int main() {
    int i, j;
    i = 3;
    j = 5;
    x(j, i, j);
    cout << i << " " << j << endl;
    return 0;
}
```

13. 다음의 프로그램의 결과는 무엇인가?

```
#include <iostream>
int funny(int &a, int b) {
    int c = a + b;
    a = b;
    b = c;
    return c;
}
int main() {
    int x = 3;
    int y = 4;
    int z = 5;
    cout << funny(x, y) << endl;
```

```
        cout << "x is: " << x << " and y is: " << y << endl;
        z = funny(x, z);
        cout << "x is: " << x << " and y is: " << y << endl;
        return 0;
    }
```

14. funny()함수에서 main()으로 돌아 가기직전의 main()과 funny()의 활성화레코드를 그리시오,
15. 흐름객체는 왜 참조로 넘겨야 하는가?
16. SimpleWindow객체는 왜 참조로 넘겨야 하는가?
17. 참조형식으로 접수하는 두개의 옹근수형인 인수를 가지고 가장 작은 절대값을 가진 수를 0으로 설정하는 함수를 작성하시오. 함수이름은 ZeroSmaller()이다.

6.10 상수파라메터

우리는 앞에서 const예약어를 리용하여 상수값을 표현하는 프로그람객체들을 정의한적이 있다. 이 객체들은 리용될수는 있어도 수정하지는 못한다.

const객체를 리용하는데는 두가지 원인이 있다. 첫째로 프로그람을 쉽게 수정하자는데 있다. 실례로 거대한 프로그람에 각이한 위치에서 나타나는 상수값을 바꿀 필요가 있다면 매개 상수를 찾아서 그것을 고쳐 주어야 한다. 거대한 프로그람에서는 이런 일이 시끄럽게 되며 시간도 걸린다. 그러면 일부 상수들은 변경하지 못할수 있다. 이렇게 되면 프로그람에서 오유로 된다. const를 리용하는 다른 원인은 다른 사람에게 객체에 대하여 추가적이면서도 유용한 정보를 제공하는것이다. const가 붙은 객체의 정의를 만날 때 그 프로그람에서는 이것을 변경시킬수 없다는것을 알게 된다. 즉 그 객체는 읽기전용이다. 이 정보는 프로그람을 리해하고 수행하는데 아주 쓸모 있다. const는 또한 파라메터선언에도 리용된다. 그 의미는 국부선언일 때와 같다. 즉 함수는 그 객체를 변경시킬수 없다. 다음의 함수를 보시오.

```
void Example(const int a, int b, int c) {
    b = a + 3;
    a = c + 5;
}
```

첫 값주기명령은 읽기가능하므로 옳지만 두번째 명령문은 옳지 않다. C++콤파일러는 이것을 오유로 인정하고 실행하지 않는다. 아래에 const파라메터를 리용한 다른 실례를 주었다.

```
void AnotherExample(const RectangleShape &R1, RectangleShape &R2) {
    R2.SetColor(Blue);
```

두번째 명령문은 r1이 const파라메터이므로 틀린다. SetColor통보는 r1의 색을 변경한다. AnotherExample()함수는 그 파라메터가 비록 참조형으로 넘어 간다고 하여도 색을 바꿀수 없다.

경험

const참조파라메터의 리용

객체가 값으로 넘어 갈 때는 모든 객체의 복사본이 만들어 지고 복사본이 함수에 넘어 간다. 큰 객체에 대해서는 프로그람이 부가처리(overhead: cpu가 체계자원을 관리하는데 드는 시간)을 증가시키며 결국 프로그람의 실현에 영향을 줄수 있다. 이러한 객체들에 대하여 효과를 얻으려면 객체를 상수형참조파라메터로서 넘겨야 한다. 객체에 참조를 넘기면 보통 복사본을 넘기는것보다

효과가 크지만 const를 달아 준다면 함수는 객체를 변경하지 않는다. 실례로 프로그람 6-10에서 FullName이라는 인수가 const참조파라메터로 넘어 가는데 이것이 아주 효과적이다.

const파라메터는 함수가 그 파라메터값을 변경할수 없다는것을 지적한다. 함수에 넘겨 지는 인수가 상수인가 아닌가에 대해서는 더이상 언급하지 않는다.

const파라메터는 함수가 파라메터의 값을 변화시킬수 없다는것을 지적하는 예약어이다. 이 파라메터는 함수에 넘겨 지는 묶음이 상수인가, 아닌가를 가리키지는 않는다. 프로그람 6-10을 보시오.

```
#include <iostream>
#include <string>
using namespace std;
void ParseName(string &FirstName, string &LastName, const string &FullName) {
    int i = FullName.find(",");
    LastName = FullName.substr(0, i);
    FirstName = FullName.substr(i + 2, FullName.size());
    return;
}
int main() {
    string Name = "Stroustrup, Bjarne"
    string FirstName;
    string LastName;
    ParseName(FirstName, LastName, Name);
    Name = FirstName + " " + LastName;
    cout << Name << endl;
    return 0;
}
```

프로그람 6-10. 이름형식을 다시 형식화하기

ParseName()함수에서 세번째 형식파라메터 FullName은 const파라메터인데 main()에서는 Name을 넘긴다. 이때 Name은 const객체가 아니다. const는 ParseName()의 파라메터들중 FullName에만 적용하는데 이것은 ParseName()함수가 FullName을 변경할수 없다는것을 지적한다. 사실 main()에서 ParseName()은 호출한 직후에 Name이 변경된다. 함수가 변경되여서는 안될 파라메터에 const를 리용하는것은 아주 좋은 쏘프트웨어기술경험이다. 이렇게 하면 코드를 보는 사람들에게 추가적인 정보를 제공하고 콤파일러가 그 객체가 의식적이든 무식적이든 변경되지 않는다는것을 자동적으로 인식하도록 한다.

주의

const변경자를 실시하는데서 콤파일러프로그람의 제한성

거의 모든 C++콤파일러들은 값주기의 왼변에 const객체가 있으면 오유통보를 낸다. 그러나 const객체가 다른 함수에도 참조형식으로 넘겨 주면 콤파일러는 경고통보를 내거나 전혀 통보하지 않는다. 다음의 실례를 보시오.

```
void foo(int &p1, int &p2) {
    p1 = p2 + p1 + 10;
    return;
```

```
}
void example(const int CValue) {
    foo(CValue, 3);
    return;
}
```

foo()함수에서 참조파라메터 p1가 변경된다. 물론 여기에는 문제가 없다. 물론 example()함수가 foo()를 호출할 때에는 CValue객체가 const참조형인데 CValue값이 변경된다. 거의 모든 콤파일러들이 이에 대하여 오유통보를 하지 않을것이다. 그러나 일부 콤파일러들은 함수에 const객체가 넘어 간다는것을 경고통보로 사용자에게 알린다.

6.11 기정파라메터

보통 함수를 호출할 때 함수가 요구하는 파라메터들의 개수를 정확히 넘겨야 한다. 실례로 ThreeArgs라고 하는 함수에 원형선언을 보자.

```
int ThreeArgs(int a, int b, int c);
```

이 함수는 아래의 코드에서 호출된다.

```
int x = ThreeArgs(1, 2);
```

콤파일러는 C에 해당한 값을 넘기지 않았기때문에 오유통보를 낸다. 일반적으로 콤파일러는 함수호출에 필요한 모든 파라메터들이 지원되지 않았다는것을 사용자에게 알려 준다. 그러나 함수가 요구하는 모든 파라메터를 넘기지 않아도 된다면 좋다. 이 기술은 기정동작을 하는 함수를 작성하려고 할 때 쓸모가 있다. 이 경우에 기정파라메터를 리용한다. 다음의 함수는 기정파라메터를 리용한 실례이다.

```
void OutputValues(ostream &out, int Value1, int Value2,
  bool DoubleSpace = false) {
    out << "Value 1 is " << Value1 << endl;
    if(DoubleSpace)
        out << endl;
    out << "Value 2 is " << Value2 << endl;
    if(DoubleSpace)
        out << endl;
    return;
}
```

여기서 OutputValue(cout , 20, 30);은 첫 3개 파라메터에 해당한 값만을 넘긴다. 네번째 파라메터가 주어 지지 않았기때문에 함수가 실행할 때 함수파라메터 DoubleSpace는 함수머리부에 정의된 값으로 된다. 따라서 우의 코드를 실행할 때 cout흐름에 출력된 결과는

```
Value 1 is 20
Value 2 is 30
```

이다. 만일 두줄공간을 출력하려면 OutputValues를 호출하고 네번째 파라메터로서 true를 넘겨야 한다. 네번째 파라메터가 주어 졌기때문에 기정값은 무시되고 넘어 간 값이 리용된다. 따라서

```
OutputValues(cout , 20, 30, true);
```

의 결과는

```
Value 1 is 20
Value 2 is 30
```

이다. 기정파라메터를 가진 함수를 정의할 때 기정파라메터는 일반파라메터의 다음에 놓여야 한다. 실례로 다음의 명령문은 틀린다.

```
void f(int x = 5, double z, int y);
```

이 규칙은 콤파일러가 추적할수 없는 파라메터를 놓쳤다는것을 인식하지 못하게 한다. 기정파라메터를 리용하여 일반화된 입력, 출력함수를 간단히 작성할수 있다.

프로그람 6-9에서 첫 파라메터로서 입력흐름을 가진 Get()함수를 작성하였다. 그 흐름에서 호출부호를 읽는다. Get()함수를 다시 작성하여 그 파라메터가 규정되지 않으면 cin흐름에서 호출코드를 읽고 또 요구하는 다른 흐름에서도 읽을수 있게 한다. 새로 작성한 Get()의 원형선언은

```
bool Get(int &d1, int &d2, int &d3, int &d4, int &d5, istream &in = cin);
```

이며 아래와 같이 Get()함수가 호출될 때

```
if(Get(d1, d2, d3, d4, d5) && Valid(d1, d2, d3, d4, d5))
    return true;
else
    return false;
```

그것은 cin흐름에서 자료를 읽는다. 파일에서 자료를 읽으려면 6번째 파라메터를 제공한다. 따라서 아래의 코드

```
ifstream fin("telcodes.dat");
if (Get(d1, d2, d3, d4, d5) && Valid(d1, d2, d3, d4, d5))
    return true;
else
    return false;
```

는 telcodes.dat파일과 접촉된 fin흐름에서 자료를 읽는다.

주의

기정파라메터는 재정의할수 없다.

C++는 기정파라메터의 재정의를 할수 없게 한다. 즉 다음의 코드는 옳지 않다.

```
void f(int x, int y=3); //원형선언
...
...
void f(int x, int y=3); { //옳지 않다.
// f의 본체
...
}
```

기정파라메터의 값은 함수의 정의나 원형선언중 어느 한 곳에서 지정하여야 한다. 원형선언이 함수의 대면부이므로 함수의 원형선언에서 기정값을 규정하기로 한다. 그러나 그와는 상반대는 경우도 만나게 된다. 때로는 원형선언에서, 때로는 함수정의에서 기정값을 규정한다.

6.12 함수파라메터의 형변환

함수호출에 대하여 생각하는 한가지 방법은 그것을 연산자처럼 생각하는것이다. 물론 함수는 많은 연산수들을 가질수 있으며 값을 만들수 있고 그렇지 않을수도 있다. 실례로 명령문

```
int Sum = Plus(a, b, c);
```

는 연산수 a, b, c에 더하기를 실시하는것으로 생각할수 있다. 이처럼 연산자에 적용하였던 많은 개념들도 함수호출에 적용한다. 실례로 다음의 원형선언과 함수호출을 보시오.

```
double ComputeInterest(double Principle, double InterestRate, int Days);
...
...
double Interest = ComputeInterest(4500, 0.075, 365);
```

ComputeIntersest()함수호출에서 첫번째 파라메터의 형은 int형인데 함수의 원형선언에는 double형으로 되여 있다. C++콤파일러에서 산수연산자의 연산수들을 적당한 형식으로 바꾸는것과 마찬가지로 파라메터를 적당한 형식으로 바꿀것이다. 우의 실례에서 콤파일러는 4500을 double로 변환하고 그 값을 넘길것이다. 마찬가지로 다음의 명령문에서

```
double Interest = ComputeInterest(500.0, 0.8, 155.8);
```

세번째 파라메터 155.8은 int로 변경된다. 따라서 Days파라메터는 155라는 값을 받는다.

6.13 함수의 다중정의

이미 C++와 같이 다른 프로그람작성언어들도 다중정의된 연산자들이 있다는것을 취급하였다. 즉 의미를 가지고 실행되는것이 무슨 연산자인가는 연산수의 자료형에 의존된다. 실례로 더하기연산자가 다중정의되였다고 하자. 그의 연산수가 옹근수라면 옹근수더하기를 실현한다. 그의 연산수가 류점수라면 류점수더하기를 실현한다. C++는 또한 함수의 다중정의도 지원한다. 즉 서로 다른 동작을 하는 같은 이름을 가진 함수들을 창조할수 있다. 함수다중정의는 서로 다른 자료형에 대하여 비슷한 과제를 실현하는 함수들을 작성하려 할 때 쓸모 있다. 6.7에 있는 다음의 Swap()를 함수다중정의실례로 리용한다.

```
void Swap(int &x, int &y) {
    int tmp = x;
    x = y
    y = tmp
    return ;
}
```

함수는 두개의 참조옹근수파라메터를 받아 그 값을 교환한다. 만일 두개의 double값을 교환할 함수가 필요할 때 함수의 다중정의가 없다면 서로 다른 이름을 가진 함수를 작성한다. 즉 SwapDouble()를 호출할수 있다. 새 이름들을 달아 주는것은 불편한 점이 있다. 함수를 다중정의하면 서로 다른 자료형에 대하여 같은 동작을 수행하는 다른 함수를 정의할 필요는 없다. 즉 파라메터는 다르고 이름이 같은 두개의 함수를 창조할수 있다. 아래의 두개의 Swap()함수를 주었다.

```
void Swap(int &x, int &y) {
    int tmp = x;
    x = y
    y = tmp;
    return;
}
void Swap(double  &x, double &y) {
    int tmp = x;
    x = y
    y = tmp;
     return;
}
```

콤파일러프로그람이 Swap()함수를 호출할 때 어느것이 호출되는가를 어떻게 결정하는가? 대답은 콤파일러는 다중정의된 연산자를 만날 때 어느 형의 연산을 실현하는가를 결정하는데 리용하던것과 비슷한 기법을 리용한다는것이다. 연산자가 다중정의되면 연산수의 형에 따라 연산이 수행된다. 함수가 다중정의되면 실제파라메터의 명세부가 어느 함수를 호출해야 하는가를 결정한다. 함수의 서명(signature)이란 그의 파라메터들의 목록이다. 콤파일러프로그람은 함수의 파라메터목록을 검사하고 실제파라메터에 가장 적합한 기호를 가진 함수를 호출한다. 실례로

```
double w = 1.2;
double z = 3.4;
Swap(w, z);
```

에서 두개의 파라메터가 다 double형이므로 double파라메터를 가진 Swap()를 호출한다. 한편 아래의 명령문

```
int i = 2;
int j = 4;
Swap(i, j);
```

은 옹근수파라메터를 가진 Swap()함수를 호출한다. 둘이상의 함수가 같은 이름을 가질 때 어느 함수를 호출하겠는가 결정하는것을 함수다중정의해결(function overload resolution)이라고 한다. 함수다중정의해결은 형식파라메터의 자료형이 실제파라메터의 자료형과 정확히 일치하는 함수일 때만 간단하다. 즉 적합한 함수가 호출된다. 파라메터형이 일치하지 않고 콤파일러가 실제파라메터를 변화시켜야 한다면 함수다중정의해결은 아주 어려워 진다. 실례로 다음의 명령문에서 Swap()의 호출은

```
double z = 2.4;
int k = 4;
Swap(z, k);
```

실제파라메터와 형식파라메터가 일치하지 않기때문에 옳지 않다. 콤파일러가 실제파라메터를 형식파라메터에 적합하도록 형변환할수 있는 두가지 방법이 존재한다. 즉 콤파일러로 첫번째 실제라메터를 옹근수로 변환할수도 있고 두번째 실제파라메터를 double형으로 변환할수 있다.

주의

표준본보기서고(STL)

STL에는 자료형에 기초하여 전문화될수 있는 유효한 함수들이 많다. 이러한 함수들을 본보기(template)라고 한다. 실례로 STL에는 6장에서 본 min과 max함수를 창조하는 본보기도 있다. 마찬가지로 서로 다른 형을 가진 교환함수의 본보기도 있다. 본보기와 그것을 리용하는 방법은 14장에서 토론한다. 우수한 C++프로그람작성자들은 STL이 제공한 특성들을 알고 그것들을 효과적으로 리용한다.

문 제

18. 함수의 서명이란 무엇인가?
19. 함수의 다중정의해결이란 무엇인가?
20. 다음의 프로그람의 결과는 무엇인가?

```
#include <iostream>
using namespace std;
void g(int i, int &j, int k = 9) {
    i= j;
    j=i-k;
    k=j*i
}
int main() {
    int i = 5;
    int j = 3;
    g(j, i, j);
    cout << i << endl;
    return 0;
}
```

21. 다음의 프로그람의 결과는 무엇인가?

```
#include <iostream>
using namespace std;
void g(int i, int &j, int k = 9) {
    i = j;
    j = i - k;
    k = j * i
}
int main() {
    int i = 5;
    int j = 8;
    g(a, b);
    cout << b << nedl;
    return 0;
}
```

22. 다음의 프로그람의 결과는 무엇인가?

```
#include <iostream>
using namespace std;
void f(int z, float b, char c) {
    cout << "a is" << b << endl;
    cout << "b is" >> a <<endl;
void f(float b,int a, char c) {
    cout << "b is" << b << endl;
    cout << "a is" << a <<endl;
int main() {
    float x = 5.0;
    f(x, i, 'd ');
    f(i, x, 'd')
    return 0;
}
```

23. 다음의 프로그람에서 오유는 무엇인가?

```
#include <iostream>
using namespace std;
void g(int i, int &j, int k = 0)
int main() {
    int a = 5;
    int b = 8;
    g(a, b);
    cout<< b << end1;
void g(int i, int &j, int k=9) {
    i = j;
    j = i - k;
    k = j * i;
}
```

6.14 재귀함수

C++를 비롯한 많은 프로그람작성언어는 문제해결을 위하여 재귀를 리용한다. 재귀(recursion)란 함수가 자기자체를 호출할수 있는 능력이다. 재귀호출과 그 리용을 실례 들기 위하여 차례곱함수의 실례를 고찰해 보자. 앞서 정의한 차례곱의 정의

$$n! = \begin{cases} 1 & n=0 \\ n\times(n-1)\times\ldots\times 1 & n\geq 1 \end{cases}$$

는 줄임부호를 리용하기때문에 수학적으로는 정확치 않다.

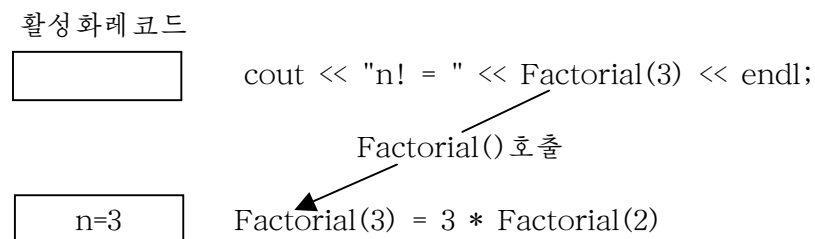
$$n! = \begin{cases} 1 & n=0 \\ n \times (n-1)! & n>0 \end{cases}$$

이 형태의 정의에서 차레곱이 또 차레곱을 정의한다는것을 알수 있다. 즉 n>0이면 n!은 n×(n-1)!이다. 재귀호출을 리용하여 차레곱을 계산하는 C++함수를 작성할수 있다.

```
int Factorial(int n) {
if (n == 0)
    return 1;
else
    return n * Factorial(n-1);
}
```

이 함수는 명백히 차레곱의 수학적정의를 반영한다. n의 값이 0이면 1이 돌려 진다. n의 값이 0이 아니면 n과 Factorial(n-1)이 돌려 진다. 재귀호출과정을 리해하려면 함수가 자기를 호출할 때 일어 나는 현상을 시각화해야 한다. 그것을 하기 위한 한가지 방법이 재귀호출시 창조되는 활성화레코드를 그리는것이다. 다음의 프로그람을 보시오.

```
#include <iostream>
#include <string>
using namespace std;
int main(){
    cout << "Please enter a positive integer: ";
    int n;
    cin >> n;
    cout << n << "! = " << Factorial(n) << endl;
    return 0;
}
```

main()이 Factorial()을 호출한후 활성화레코드는 다음과 같다.

Factorial()에서 n은 3이므로 **if**명령문의 **else**부분이 실행된다.

```
return n * Factorial(n-1);
```

else부분은 Factorial()을 다시 호출하며 이때 실제파라메터값은 2이다. Factorial()함수의 호출에서

n이 2이므로 if명령문의 else부분이 다시 실행된다.

```
return n * Factorial (n-1)
```

이때 실제파라메터의 값은 1이다.

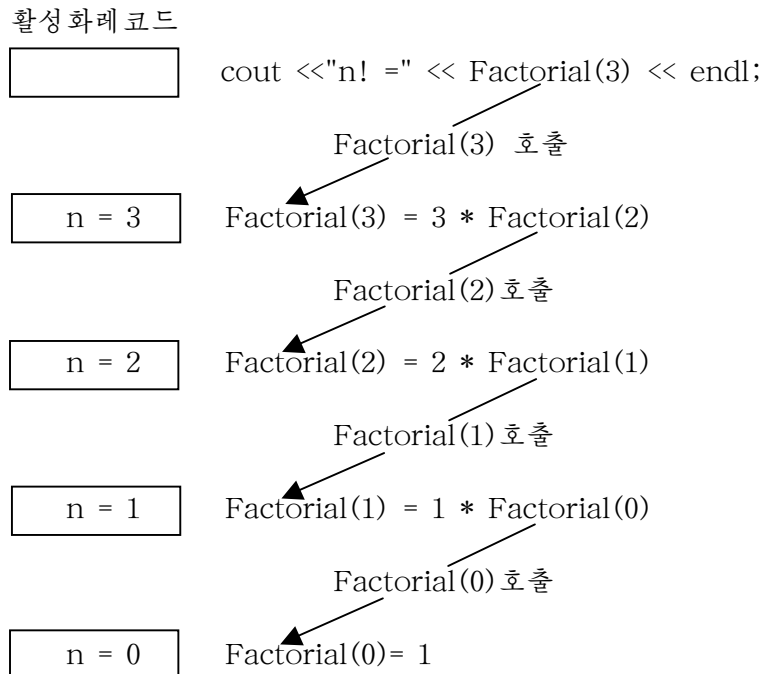



그림 6-8. Factorial()함수의 재귀호출을 위한 활성화레코드

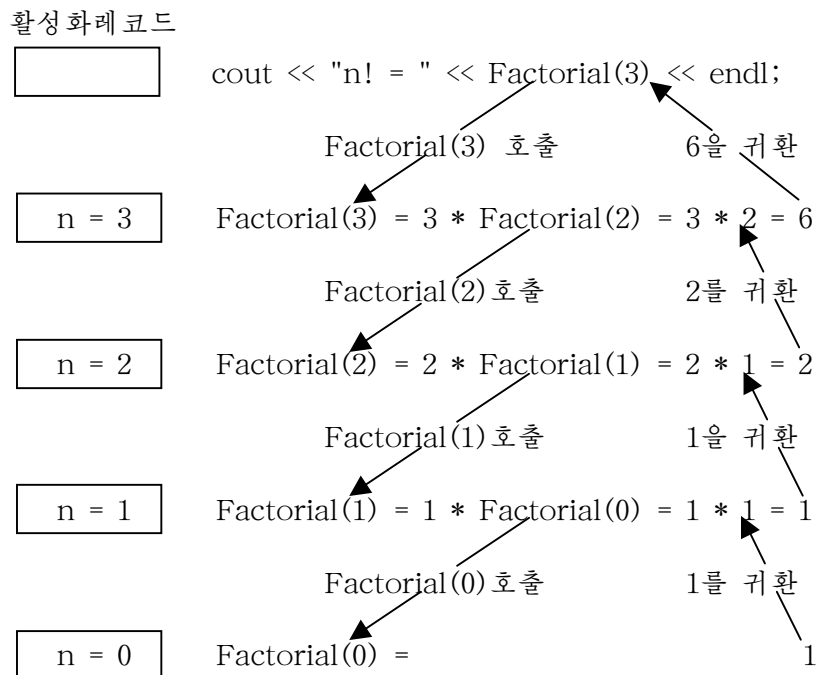



그림 6-9. Factorial()함수의 재귀적호출의 끝내기

그림 6-8에 보여 준것처럼 처리는 n이 0일 때까지 계속된다. 이때 if조건식이 true로 평가되며

```
return 1;
```

가 실행된다. 이 명령문은 재귀호출이 더이상 진행되지 않도록 한다. 재귀호출이 끝나면 함수를 호출한 곳으로 값이 돌려 진다. 되돌림값이 호출에 대응된다. 재귀호출끝처리는 그림 6-9에 실례로 들었다. 되돌림값에 대응하여 n차례곱은 계산되여 다음준위에로 되돌아 간다. 이 처리는 main()함수에서 Factorial()에로 호출이 3!값을 가지고 main()에로 돌아 갈 때까지 계속된다. Factorial()에 대한 첫 호출이 main()에로 돌아 갈 때 출력명령문의 결과는 다음과 같다.

```
3! = 6
```

재귀호출이 가지는 우점은 어떤 수학공식을 간결하게 표시할수 있다는것이며 그의 C++실현이 간단하다는것이다. 재귀함수는 일반적으로 두개의 부분을 가진다.

- 단순한 파라메터를 가진 재귀호출
- 재귀호출을 중지하는 완료부분

차례곱코드에서 재귀호출과 완료부분은 **if-else**명령문의 두 기능부분에 위치한다.

```
        if (n == 0)          } 완료부분
            return 1;
재귀부분 { else
            return n * Factorial(n - 1);
```

x^x을 계산하는 재귀함수를 개발하여 보자. 제곱함수의 원형선언은

```
int Power(int x, int n);
```

이다. 모든 x에 대하여 x^0은 1이다. 이 사실은 재귀함수의 완료부분으로 된다. 또한 $x^n=x \times x^{n-1}$이다. 이것은 재귀호출부분으로 된다. N이 작아 지기때문에 재귀호출을 완료해야 하는가를 관찰하시오. 제곱함수의 재귀호출부분코드작성은 간단하다.

```
int Power(int x, int n) {
    if (n == 0)
        return 1;
    else
        return x * Power(x, n-1);
}
```

재귀호출의 마지막실례로서 n번째 피보나치수를 계산하는 함수를 작성하자. 4장에서 이 수렬을 고찰하였다. 피보나치수렬이 F1, F2, …, Fn앞에 F1=F2=1이고 다음원소는 앞선 두 원소의 합이다. 수렬에서 첫 8개 수는 1, 1, 2, 3, 5, 13, 21이다. 피보나치수렬은 이딸리아수학자 레오나르드 피사노가 1202년에 발견하였는데 그는 토끼가 리상적인 환경에서 어느 정도 빨리 번식하는가를 조사하고 있었다. 그는 토끼 한쌍(수컷과 암컷)이 또 다른 한쌍의 토끼를 낳는다고 가정하였다. 토끼가 한달후면 성적발육이 완성되며 새끼 배는 기간은 한달이다. 따라서 토끼 한쌍은 두달후부터는 매달 한쌍의 토끼를 낳는다고 가정하면 1년후에는 토끼가 몇마리이겠는가(물론 토끼는 죽지 않는다)? 이것을 수동적으로 줄수 있다. 첫달 마지막에는 한쌍밖에 없다. 두달 마지막에는 다른 쌍이 태여나 두쌍이 된다. 석달 마지막에는

원래 있던 쌍이 다른 쌍을 낳고 두번째 쌍은 번식활동이 시작되므로 3쌍이 된다. 넉달 마지막에는 초기쌍이 다른 쌍을 낳고 두번째 쌍도 다른 쌍을 낳아 5쌍이 된다. 다섯달 마지막에는 3쌍이 된다. 여섯달후에는 13쌍이 된다. 즉 수렬을 구성하면

1, 1, 2, 3, 5, 8, 13, 21, 34...

이다. 즉 매달 토끼쌍수는 앞서 두달의 토끼쌍수의 합이다. 12달후에는 144쌍의 토끼가 된다. n번째 피보나치수의 수학적정의이다.

$$Fn = \begin{cases} F_n = 1 & n = 1\text{인 경우} \\ F_n = 1 & n = 2\text{인 경우} \\ F_n = F_{n-1} + F_{n-2} & n > 2\text{인 경우} \end{cases}$$

이전에 정의한 재귀함수보다는 조금 어렵지만 같은 방법을 리용할수 있다. 이전의 재귀함수형식은

```
if(termination code satisfied)
    return value;
else
    make simpler recursive call;
```

였는데 이 형식은 피보나치수렬에 대하여서도 가능하다. 함수의 코드는

```
int Fibonacci (int n) {
    if (n <= 2)
        return 1;
    else
        return Fibonacci(n-1) + Fibonacci(n-2);
}
```

이다. 현재코드와 이전의 재귀함수코드와의 기본차이가 현재코드에는 두개의 재귀호출이 참가한다는것이다. 매개 재귀호출이 작아 지는 n을 넘기기때문에 결국에는 재귀호출이 끝난다는것을 알고 있다. 그림 6-10에 F_5를 계산할 때 이 코드에 의하여 작성된 재귀호출을 실례 들어 보여 주었다. 피보나치수렬은 자연계의 많은 대상을 포함하기때문에 흥미 있는것이다. 실례로 꿀벌의 생식활동과 나무의 가지수, 꽃에 있는 꽃잎수를 모형화한다(실례로 미나리아재비에는 5개의 꽃잎이, 개미취에는 21개의 꽃잎이 있으며 꽃잎이 34, 55, 89개인 들국화도 있다).

```
if (termination code satisfied)
    return value;
else
    make simpler recursive call;
```

련습에서 피보나치수렬의 다른 리용분야를 실례로 들기도 한다.

```
int Fibonacci(int n) {
    if ( n <= 2)
        return 1;
    else
```

```
        return Fibonacci(n-1) + Fibonacci(n-2);
    }
```

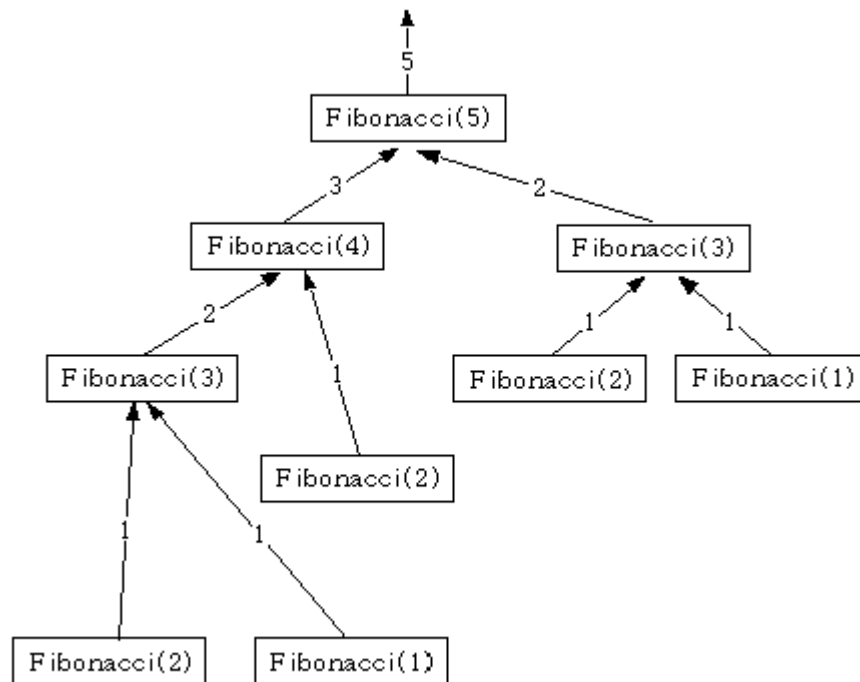



그림 6-10. F_5의 계산

문 제

24. 다음의 프로그램의 출력결과는 무엇인가?

```
#include <iostream>
using namespace std;
int f(int n) {
    if(n <= 1)
        return n;
    else
        return f(n-1) + f(n-2);
}
int main() {
    cout << f(4) << endl;
    return 0;
}
```

25. 2개의 문자렬인수 s와 r를 리용하는 Reverse()재귀함수를 작성하시오. Reverse()함수는 문자렬 s에 있는 문자들을 반전시켜 문자렬 r에 배치하는데 재귀호출을 리용한다.
26. 한개의 옹근수를 인수로 받는 재귀함수 PrintNumber()를 작성하시오. PrintNumber()함수는 그의 인수값을 한번에 한 문자씩 출력하는데 재귀호출을 리용한다.

6.15 가격구간별 증권도표의 현시

진지한 투자가들은 증권시장의 움직임을 파악하는데 많은 시간을 보낸다. 그들이 사용하는 많은 쏘프트웨어도구들은 도형적인 수단으로 시장의 움직임을 더욱 명백히 알수 있도록 정보를 표현한다. 문제는 주어 진 증권에 대하여 주간의 가격구간을 그라프적으로 어떻게 표시하는가 하는것이다. 문제서술은 주별 증권리자가격이 포함된 파일에서 가격구간문제를 해결한다. 파일이름은 사용자에 의하여 입력된다(파일이름과 그안에 있는 가격은 검사되여야 한다).

파일에서 가격은 한조씩 얻는다. 한조는 주간 증권가격의 최고값과 최소값을 표현한다. 모든 조들이 입력되면 표식된 그라프상에 선이 그려 진다. 실례로 다음의 표본입출력동작은 증권파일이름을 추출하는 동작이다.

please enter file to be processed:stock.dat

여기서 stock.dat파일에는 다음의 값들이 들어 있다.

2 4
1 5
4 6
4 8
5 9
3 8

그림 6-11에 그라프를 주었다. 그라프에는 축마다 표식이 붙어 있는데 하나는 증권자료의 주번호이고 다른 하나는 증권가격의 최고값을 주었다. 주별자료를 적당히 표현하는것은 문제해결의 기본고리이다. 순수한 객체지향설계에서는 주별 최저/최고값을 다 표현하는 객체를 참조해야 한다. 그리고 그 객체에 유효성검사와 대응한 가격구간을 그라프로 그리는 성원함수를 작성해야 한다. 7장시작에서 이러한 객체를 정의하기 위한 클라스기구를 소개한다. 2장의 련습문제에 이것을 해결하기 위한 문제가 하나 있다. 그러나 지구별 최저/최고값을 표현하는 2개의 독립인 객체가 필요하다. 이 두개 값과 표시창이 주어 지면 가격구간을 RectangleShape객체를 리용하여 표시하기때문에 아주 간단하다.

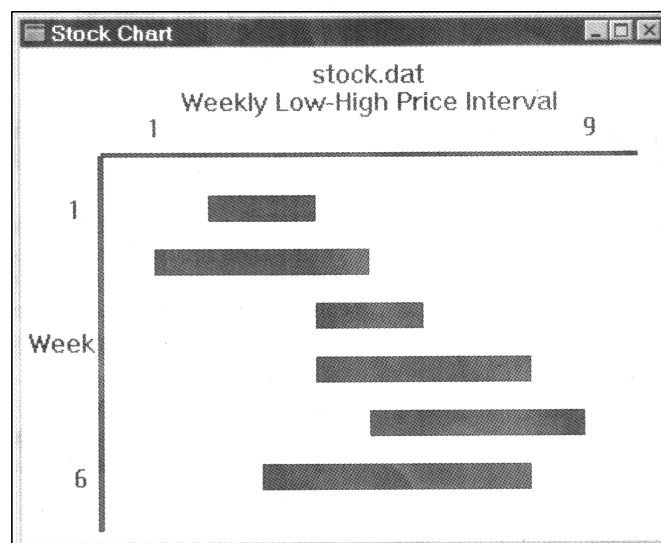



그림 6-11. stock.dat파일의 가격구간별 도표

x축에는 모든 주에 대하여 본 최대증권가격을 기록한다. y축에는 증권자료의 두 번호를 기록한다. 이 두개 값을 계산하는것은 간단하다. 최대가격은 현재까지 최고가격과 현재주간의 높은 가격을 반복적으로 비교하여 필요하면 최대값을 변경한다. 비교를 위하여 6.3에서 취급한 Max()함수를 리용한다. 주번호는 주별로 입력할 때마다 한번씩 계수기를 증가시킨다.

이와 같은 분석에 기초한 문제해결알고리듬은 아래와 같다.

단계1: 파일이름입력을 재촉하고 추출

단계2: 추출된 파일이름을 가진 입력흐름을 정의

단계3: 주별계수기를 설정하고 모든 증권가격을 높이 설정

단계4: 주별 최저/최고가격을 입력할 때

단계4.1: 주별 계수기증가

단계4.2: 증권최고가격과 현재 입력한 주별 최고가격을 비교하고 필요하면 증권최고가격을 갱신

단계4.3: 주별 가격구간을 그라프로 그리기

단계4.4: 4단계를 반복

단계5: x축표시

단계6: y축표시

알고리듬은 프로그람을 서술하기 위한 기본단계들을 묘사하고 있다. 그 단계들이 다음의 코드토막에 반영된다.

```
string StockFile = GetFilename();
ifstream fin(StockFile.C_str());
if (! fin) {
    cerr << " Cannot Open:" << StockFile << endl;
    exit(1);
}
float StockHigh = 0;
int WeekNbr = 0;
int WeeklyHigh;
int WeeklyLow;
while (ReadStockInterval(fin, StockFile, WeeklyLow,
  WeeklyHigh, WeekNbr) {
    ++WeekNbr;
    StockHigh = Max(StockHigh, WeeklyHigh);
    ChartWeek(WeekNbr, WeeklyLow,  WeeklyHigh);
}
DrawXAxis(StockFile, StockHigh);
DrawYAxis(WeekNbr);
```

코드토막에서 보다 구체적인 알고리듬부분은 본질에 있어서 우에서 아래로 설계되는 함수에 있다. 이 설계방법에서 사용자는 문제해결을 위하여 실행되여야 할 기초과제를 결정한다. 또한 이 과제들이 어

떻게 결합하여 호상작용해야 하는가를 결정한다. 필요하다면 기초과제를 모든 동작이 잘 리해되고 실현하기 쉬울 때까지 분해하지 않는다. 실례에서는 함수에 6개의 과제를 분해한다.

- GetFilename(): 가격구간을 포함하고 있는 파일의 이름을 얻는다.
- ReadStockInterval(): 주별 증권가격을 읽고 검증한다.
- Max():가장 높은 증권가격을 갱신하게 한다.
- ChartWeek():현재의 주별 가격구간을 표시한다
- DrawXAxis(): x축을 그리고 표식을 붙인다.
- DrawYAxis(): y축을 그리고 표식을 붙인다.

문제해결을 위하여 함수를 개발할 때 함수는 보통 목적에 따라 파일들을 묶어서 배치한다. 즉 함수파일들은 비공개서고들이다. 같은 시각에 쏘프트웨어기사는 또한 이 함수파일을 위한 머리부파일을 창조할수 있다. 비공개서고가 필요되면 련결된 머리부파일은 보통함수를 요구하는 프로그람파일의 시작위치에 포함한다. 실현부파일이 아니라 머리부파일이 포함된다. 실현부파일은 꼭 한번 콤파일되여 비공개서고를 리용하는 응용프로그람을 번역할 때 필요한 경우에만 련결된다. 그 결과 응용프로그람의 콤파일이 빨라 진다.

가격도표프로그람은 2개의 실현부파일 stock.cpp와 utility.cpp 그리고 머리부파일 utility.h를 포함한다.

실현부파일 stock.cpp에는 ApiMain()이 있다. 사실 창문화특성과 관련한 일부 코드를 제외하고는 이전 코드와 ApiMain()의 본체는 같다. 3장에서 취급한것처럼 ApiMain()은 프로그람의 시작함수이다. stock.cpp의 내용을 목록 6-1에 주었다.

ApiMain()의 정의와 함께 stock.cpp에는 필요한 표준서고, 머리부파일들과 국부서고머리부파일인 ezwin.h와 utility.h가 있다. Ezwin.h에는 기초적인 도형함수들의 원형이 들어 있다. utility.h에는 ApiMain()함수가 리용하는 6개의 함수원형이 들어 있다.

목록 6-2는 utility.h의 내용이다. 함수원형들은 전처리기지령안에 겹쳐 들어 있다. 이 지령들은 함수의 원형들이 매 번역마다에서 한번만 선언되도록 한다. 프로그람작성자가 정의한 서고용머리부파일은 보통 원천파일의 시작위치 즉 표준서고용머리부파일포함지령의 다음위치에서 포함된다.

utility.cpp의 내용을 목록 6-3에서부터 6-5사이에 주었다. 목록 6-3은 utility.cpp의 초기부분이다. 머리부파일 utility.h가 포함되였는데 이것은 함수정의가 실지로 정확한 대면부를 가지는가를 확인하고 그렇지 않으면 오유통보가 발생하도록 한다. 파일포함지령다음에 여러개의 상수가 정의된다. 상수는 축과 막대기도표의 특성 그리고 그라프가 창문의 초기위치로부터 어느 정도 편위되여야 하는가를 서술한다.

목록 6-1. 유효범위문제에 관한 프로그람

```
#include <iostream>
#include <fstream>
#include <string>
#include <stdlib.h>
#include "ezwin.h"
#include "utility.h"
using namespace std;
int ApiMain() {
```

```
    string StockFile = GetFileName();
    ifstream fin(StockFile.C_str());
    if(! fin) {
    cerr << " Cannot open:" << StockFile << endl;
    exit(1);
    }
    SimpleWindow W(" Stock Chart", 12, 9);
    W.Open();
    float StockHigh = 0.0f;
    int WeekNbr = 0;
    int WeeklyHigh;
    int WeeklyLow;
    while (ReadStockInterval(fin, StockFile, WeeklyLow,
      WeeklyHigh, WeekNbr)) {
        ++WeekNbr;
        CharWeek(W, WeekNbr, WeeklyLow, WeeklyHigh);
        StockHigh = Max(StockHigh, WeeklyHigh);
    }
    DrawXAxis(w, StockFile, StockHigh);
    DrawYAxis(w, WeekNbr);
    Cout << "Type a character followed by a\n"
      << "return to remove the graph and exit" << endl;
    char Anychar;
    cin >> AnyChar;
    W.Close();
    return 0;
}
```

함수 GetFilename(), Valid(), Max()를 목록 6-3 에서 정의하였다. 이 함수들은 간단하거나 이미 취급하였으므로 설명을 달지 않는다.

목록 6-3에는 또한 **void**함수 ChartWeek()가 있다. ChartWeek()함수는 4개 파라메터를 가진다. 첫번째 파라메터 W는 그라프를 가진 SimpleWindow이다. 이 파라메터는 함수가 현재주의 가격구간을 배치하여 창문을 변경하기때문에 참조형파라메터로 되여야 한다. 마찬가지로 DrawXAxis(), DrawYAxis()함수는 SimpleWindow객체에 대하여 참조형파라메터를 가진다.

목록 6-2. utility.cpp의 함수들의 원형이 들어 있는 utility.h

```
#ifndef UTILITY_H
#define UTILITY_H
#include <iostream>
#include <string>
```

```
using namespace std;
string GetFileName();
bool Valid(float low, float high)
void CharWeek(SimpleWindow &w, int week, float low, float high);
void DrawXAxis(SimpleWindow &w, const string &stockFile, float StockHigh);
void DrawYAxis(SimpleWindow &w, float WeekNbr);
int Max(int a, int b);
bool ReadStockInterval(istream &file, const string
  &FName, int &Low, int &High, int WeekNbr);
#endif
```

3개의 float형값파라메터들인 Week, Low, High는 주번호와 가격구간의 끌점들을 표현한다. 함수는 주별, 증권가격구간을 표현하는 막대기를 표시한다. 가격구간막대기의 중심의 x자리표는 도표의 x축상수 ChartXoffset와 low와 High의 평균값을 합한것이다. 막대기의 중심 y축자리표는 Week값을 도표의 y축상수인 ChartYoffset로 밀기하는것이다. 막대기의 길이는 High-Low이다. 막대기의 색갈은 Barcolor상수가 지정한다. RectangleShape객체 Bar가 이런 특성으로 정의되면 Draw()성원함수는 증권가격구간을 표시한다.

목록 6-3에는 또한 ReadStockInterval()함수가 있다. 6.8에서 설명하였지만 프로그람의 입력함수는 각이한 함수에 배치된다. ReadStockInterval()함수의 첫번째 인수는 자료를 읽는 흐름이다. 6.8에서 본 ReadValues()함수와 비슷하게 입력흐름이 cin이라면 함수는 사용자에게 입력을 진행할것을 알린다. ReadStockInterval()함수는 파일에서 자료를 읽도록 리용할수 있다. 언제나 그러하듯이 입력을 할 때 자료의 유효성을 검사한다.

목록 6-4에서는 void형함수 DrawXAxis()를 정의하는데 이 함수는 x축과 그의 표식을 표시한다. 표시를 완수하기 위하여 함수는 이전에는 리용하지 않았던 두개의 클라스 Label과 ostringstream을 사용한다.

Label클라스는 또 다른 도형클라스이다. Label객체 SimpleWindow에 있는 통보를 표시하는데 리용된다. Label객체를 초기화할 때는 표식이 붙은 SimpleWindow객체와 그 창문의 위치 그리고 희망하는 통보만을 규정하면 된다.

ostringstream은 iostream계층의 한 부분이다. ostringstream클라스의 객체는 기억기흐름이라고 하는데 그것은 출력이 감시기나 파일에 넘어 가는것이 아니라 문자렬과 같은 기억기완충기에 저장하기때문이다. ostringstream객체에로의 출력이 완성되면 구축된 문자렬이 ostringstream성원함수 str()에 의하여 호출될수 있다. ostringstream클라스는 표준머리부파일 sstream에 선언된다. 이 머리부파일에는 또한 기억기입력흐름클라스 istringstream과 입력과 출력을 다할수 있는 기억기클라스 ostringstream이 선언되였다.

목록 6-3. utility.cpp의 초기부분

```
#include <iomanip>
#include <sstream>
#include <string>
```

```
#include "rect.h"
#include "label.h"
#include "utility.h"
using namespace std;
//상수정의
const float AxisThickness = 0.1f;
const color AxisColor = Blue;
const float BarThickness = 0.5f;
const color BarColor = Blue;
const float ChartXoffset = 1.5f;
const float ChartYoffset = 2.0f;
// GetFileName(): 파일이름을 추출한다.
string GetFileName() {
    cout << "please enter file to be processed:";
    string s;
    cin >> s;
    return s;
}
// Valid(): 주간가격값을 확인한다.
bool Valid(float low, float high) {
    return (0 <= low) && (low <= high);
}
//Max:두개 파라메터에서 큰 값을 얻어 낸다.
int Max(int a, int b) {
    if (a<b)
        return b;
    else
        return a;
}
//ChartWeek(): 현재주의 간격값을 현시한다.
void ChartWeeks(SimpleWindow &w, int Week, float Low, float High) {
    float x = ChartXoffset + (Low + High)/2.0;
    float y = ChartYoffset + Week;
    float Length = High-Low;
    RectangleShape Bar(W, x, y, BarColor, Length, BarThickness);
    Bar.Draw();
    return;
}
//ReadStockInterval(): 주간의 높고낮은 가격값을 읽어 낸다.
```

```
bool ReadStockInterval(istream &fin, const string &Filename, int &Low, int &High,
  int Week) {
    if (fin == cin)
        cout << "enter the low and high stock price";
    fin >> Low >> High;
    //자료가 거짓을 되돌리는 경우
    if (! fin)
        return false;
//유효한 자료인가를 검사한다.
if (! Valid(Low, High) {
    cerr << Filename << ":Bad data for week"
      << Week +1 << endl;
    exit(1);
    }
return true;
}
```

목록 6-4. utility.cpp에서 DrawXAxis()함수

```
void DrawXAxis(SimpleWindow &W, const string &StockFile, float MaxX) {
    float AxisLength = MaxX +1;
    float CenterX = ChartXOffset + MaxX/2.0+0.5;
    float CenterY = ChartYOffset;
    RectangleShape Axis(w, CenterX, CenterY, AxisColor,
      AxisLength, AxisThickness);
    Axis.Draw();
    float Filenamex = CenterX;
    float Filenamey = ChartYOffset/4.0;
    Label FileName(W, FilenameX, FilenameY, StockFile);
    Filename.Draw();
    float LegendX = CenterX;
    float LegendY = ChartYOffset/2.0;
    Label Legend(W, LegendX, LegendY,
      "Weekly Low-High Price Interval");
    Legend.Draw();
    float LowX = ChartXoffset +1;
    float LowY = ChartYoffset/2.0+0.5;
    Label Low(w, LowX, LowY, "1");
    Low.Draw();
    float HighX = ChartXoffset + MaxX;
```

```
    float HighY = ChartYoffset/2.0+0.5;
    ostringstream HighValue;
    HighValue << setw(3) <<MaxX;
    Label High(w, HighY, HighValue.str());
    High.Draw();
    return;
  }
```

DrawXAxis()함수는 축의 길이를 정의하는것으로부터 시작된다. 미학적인 목적으로부터 축은 MaxX파라메터보다 큰 단위이며 MaxX파라메터는 가격구간막대기표시에 리용되는 x축최대값이다. x, y자리표 CenterX와 CenterY는 그다음 정의한다. 가격구간막대기의 표시로서 축의 중심자리표는 창문의 초기위치로부터 우로 밀기된다. 이 값들이 정의되면 축은 RectangleShape객체로서 표현되고 그려진다.

string파라메터 Filename으로 처리되는 파일의 이름과 축에 표시할 문자들이 Label객체를 리용하여 표시된다. 이 2개의 Label객체위치는 x축의 중심자리표에 관계된다.

DrawXAxis()함수는 x축상에 대하여 최저/최대값을 표식으로 구축하고 표시한다. 최대값이 1이므로 축의 왼쪽끝을 표시하는 Lagbel객체 Low를 정의하기는 쉽다. 그러나 축의 오른쪽을 표식하는것은 간단치 않다. 왜냐하면 수값을 문자렬로 자동변환할수 없기때문이다. MaxX의 값을 ostringstream객체 HighValue에 삽입하여 문자렬을 표현할수 있다. setw(3)조종기호는 마당너비를 최소한 0으로 설정한다. 이와 함께 HighValue, str()을 호출하여 MaxX에 대한 적당한 문자렬을 만든다.

목록 6-5에서 void함수 DrawYAxis()를 정의하였다. 그의 조작은 DrawXAxis()와 같으므로 더 언급하지 않는다.

목록 6-5. utility.cpp에서 DrawYAxixs()함수

```
//DrawYAxis(): 표식을 가진 y축을 현시한다.
void DrawYAxis(SimpleWindow &W, float MaxY) {
    //축을 그린다.
    float CenterX = ChartXOffset;
    flaot CenterY = ChartYOffset + MaxY/2.0 + 0.5;
    float AxisLength = MaxY + 1;
    RectangleShape Axis(W, CenterX, CenterY, AxisColor,
      AxisThickness, AxisLength);
    Axis.Draw();
    //값을 현시
    float LegendX = ChartXOffset/2.0;
    float LegendY = CenterY;
    Label Legend(W, LegendX, LegendY, "Week");
    Legend.Draw();
    //축의 낮은 표식을 현시
```

```
        float LowX = ChartXOffset/2.0 +0.25;
        float LowY= ChartYOffset +1;
        Label Low(W, LowX, LowY, "1");
        Low.Draw();
        //축의 높은 표식을 현시
        float HighX = ChartXOffset/2.0 + 0.25;
        float HighY = ChartYOffset + MaxY;
        ostringstream HighValue;
        HighValue << setw(3) << MaxY;
        Label High(W, HighX, HighY, HighValue.str());
        High.Draw();
        //동작완료
        return;
}
```

콤퓨터의 력사

대중수산기

1900년부터 1930년까지 기계식계산기는 속도가 빠르고 더 높은 능력을 가지게 되였다. 마찬가지로 홀러리스가 발견한 착공카드장치는 끊임없이 개선되였고 리용분야도 넓어 졌다. 1930년까지 범용콤퓨터에 대한 바베지의 환상은 전진하지 못하였지만 1930년부터 비약적인 전진을 이룩하였다. 참으로 새로운 발견이 너무도 빨리 일어 나 력사가들은 누구의 생각이고 누구의 창안품인지를 결정하는데 많은 시간을 바쳤다.

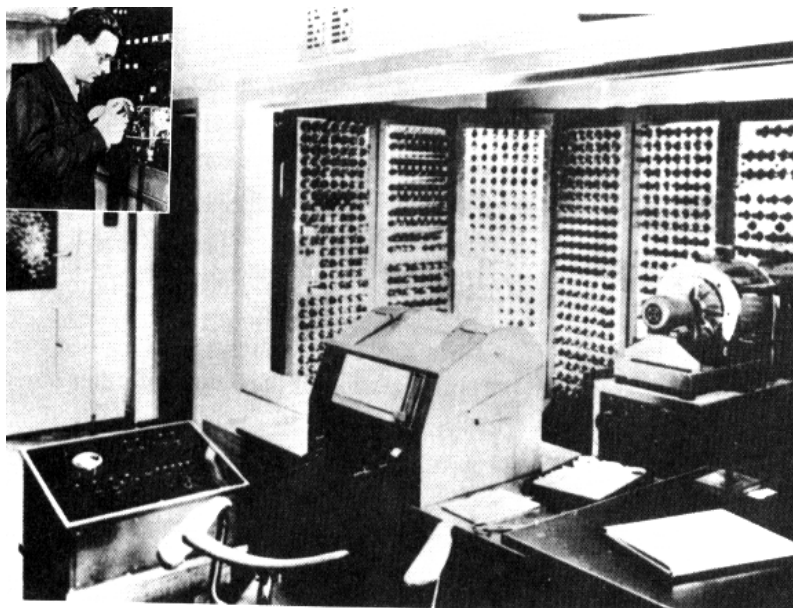



그림 6-12. 콘라드 쥬스와 Z4 기계

2가지 명쾌한 생각이 콘라드 쥬스에게 떠올랐다. 쥬스는 도이칠란드기술자인데 거치장스러운 기술계산을 싫어 하였다. 그런 계산을 쉽게 하기 위하여 쥬스는 대중수산기를 개발하리라 결심하였다. 이전에 기구들은 10진수체계를 리용하였다. 쥬스는 콤퓨터기구가 2진수체계를 리용한다면 훨씬 간단할것이라는것을 알게 되였다. 오늘날의 현대적인 고속콤퓨터들은 2진수체계를 리용한다. 쥬스의 두번째 공로는 기계식치차대신에 기계식스위치를 리용한것이다. 오늘의 모든 기계들은 스위치를 리용한다.
쥬스의 첫번째 기계를 Z1이라고 하였다. Z1은 혁신적인 기계일뿐아니라 처음으로 나온 《가정용》콤퓨터였으므로 주목할만하다. 쥬스는 베를린에 있는 자기 부모들의 집에서 Z1을 만들 때 얻은 지식과 경험을 가지고 Z2이라는 보다 큰 기계를 만들었다. Z2의 구별되는 특징은 전자기계식수산기였다는것이다. 이 기계에서는 전자식교환기가 기계식스위치를 대신하였다. 이것은 콤퓨터의 속도를 증가시켰다. 2차대전기간 쥬스는 자기의 설계를 계속 정립하여 Z3과 Z4를 만들었다. 전쟁이 끝난 다음 Z4(그림 6-12) 는 스위스에 있는 련방기술협회(Federal Polytechnical Institute:ETH)에 설치되였으며 1955년까지 리용되였다. 쥬스는 그후 소형콤퓨터회사를 설립하였다.

6.16 알아 둘 점

✓ 함수는 모듈식프로그람작성과 쏘프트웨어를 재리용하도록 하는 기구이다.
✓ 가장 단순한 함수는 수학의 함수와 비슷한 방법으로 파라메터를 가지며 식에서 리용되는 값을 계산하고 되돌린다.
✓ 명령문블로크는 대괄호안에 있는 명령문들의 목록이다.
✓ 겹싸인 블로크는 다른 명령문블로크에 포함된 명령문블로크이다.
✓ 함수를 정의하기 위하여 프로그람작성자는 그의 대면부와 동작을 완전히 규정하여야 한다. 동작은 함수본체에 일어 난다. 함수본체는 명령문블로크이다.
✓ return명령문은 호출된 함수에서 호출하는 함수에 값을 넘겨 준다.
✓ 국부객체는 명령블로크안에 정의된 객체이다.
✓ 전역객체는 함수대면부나 함수본체밖에 정의된 객체이다.
✓ 자기 과제를 완수하기 위하여 함수는 국부 혹은 전역객체를 사용할수 있으며 지어는 다른 함수까지도 리용할수 있다.
✓ 함수를 호출하기 위하여 프로그람작성자는 정확한 자료형을 가진 실제파라메터를 주어야 한다. 실제파라메터가 형식파라메터의 자료형과 맞지 않으면 콤파일러는 실제파라메터를 정확한 형식으로 변환하게 된다 .
✓ 활성화레코드(activation record)는 함수가 그의 파라메터들과 국부객체를 보관하는데 리용하는 기억기이다.
✓ 함수를 호출할 때마다 새로운 활성화레코드가 창조된다.
✓ 전형적인 프로그람작성방법은 여러가지 파일을 작성하는것이다. 그 파일들은 개별적으로 콤파일되고 서로 련결되여 실행가능한 프로그람으로 된다.
✓ 머리부파일은 흔히 실현부파일에 정의되는 함수의 원형을 선언하는데 리용된다. 머리부파일을 리용하여 실현부파일에 정의된 함수는 다른 실현부파일에서 호출될수 있다.
✓ 이름은 그와 관련된 선언이 서로 다른 블로크에서 정의되는한 재리용될수 있다.
✓ 실현부파일에 있는 전역객체의 참조는 전역객체가 extern명령문을 리용하여 정의되거나 선언될것을

요구한다.

- ✓ 전역객체는 프로그람의 전역유효범위에서 오직 한번만 정의될수 있다.
- ✓ 우에서 아래에로 진행되는 설계에서, 사용자는 수행하여야 할 기초과제들과 이러한 과제들이 어떻게 대면하고 호상 작용하는가를 결정한다. 필요하다면 기초과제는 모든 동작이 다 리해되고 쉽게 실현될수 있을 때까지 분해된다.
- ✓ sstream서고는 문자렬완충기와 같이 동작할수 있는 기억기흐름을 제공한다.
- ✓ 삽입렬을 통하여 ostringstream객체는 다른 객체의 값을 문자렬로 표현할수 있다.
- ✓ 재귀호출은 함수가 자기자체를 호출하는것이다 . 재귀함수는 두개의 부분을 가져야 한다. 재귀호출이 끝나는 완료부분과 보다 단순한 파라메터를 가진 재귀호출부분이다.
- ✓ C++는 2가지 형식의 파라메터를 지원한다. 즉 값파라메터와 참조파라메터를 지원한다.
- ✓ 파라메터가 값으로 넘어 갈 때는 객체의 복사본이 함수에 넘어 간다. 함수에서 파라메터를 변경하면 객체의 원시값이 변하지 않고 그의 복사본의 값이 변한다.
- ✓ 참조파라메터를 사용할 때는 객체의 복사가 아니라 원시객체의 참조가 넘어 간다. 호출된 함수가 파라메터를 변경하면 원시객체가 변한다.
- ✓ 참조파라메터를 리용하는 한가지 리유는 함수에서 원시객체를 변경해야 하는 경우이다. 실례로 함수가 여러개의 값을 되돌릴 경우에 이 파라메터는 참조형이여야 한다.
- ✓ iostream객체가 함수에 넘어 갈 때 출력 혹은 입력조작은 흐름을 변경한다. 따라서 흐름객체는 참조파라메터로 넘어 가야 한다.
- ✓ 참조파라메터를 리용하는 리유는 또한 효과성이다. 객체가 값으로 넘어 갈 때 객체의 복사본이 넘어간다. 객체가 크다면 그의 복사본을 만드는것은 실행시간과 기억공간측면에서 품이 들수 있다. 따라서 크기가 크거나 크기를 알지 못하는 객체들은 참조로 넘어 간다. const변경자를 리용하여 객체가 수정되지 못하게 할수도 있다.
- ✓ const변경자가 파라메터선언에 적용되면 함수가 그 객체를 변경할수 없다. 함수가 객체를 수정하려고 하면 콤파일러는 오유를 낸다.
- ✓ C++의 기정파라메터기구는 함수호출시 파라메터값이 제공되지 않아도 기정값을 가지고 처리하도록 하는 능력을 제공한다. 따라서 함수를 호출할 때 필요한 파라메터만을 라렬한다.
- ✓ 함수를 최대한 효과적으로 리용하자면 서고함수에 대하여 될수록 일반적인 함수를 만들어야 한다. 함수는 필요한 경우에 필요이상의 파라메터들을 접수할수 있다.
- ✓ 기정파라메터들은 마지막파라메터들만 지정할수 있다.
- ✓ 함수의 파라메터목록은 함수의 서명으로 된다.
- ✓ 함수다중정의는 같은 이름을 가진 2개이상의 함수의 정의이다.
- ✓ 콤파일러는 요구하는 서명을 가진 함수를 호출하여 다중정의된 함수들의 호출을 해결한다.

참고할 책

마르가레트 엘리스와 쟈르너 스트로우스트라프의 The Annotated C++ Reference Manual에 함수다중정의해결에 대한 완전한 론의가 들어 있다.

련습문제

6.1 이름을 재리용하는 목적은 무엇인가?

6.2 함수를 호출할 때 무엇이 일어 나는가?

6.3 함수원형선언은 함수정의와 어떻게 다른가?

6.4 활성화레코드란 무엇인가?

6.5 실제파라메터와 형식파라메터사이의 차이점은 무엇인가?

6.6 국부객체란 무엇인가?

6.7 전역객체란 무엇인가?

6.8 형식파라메터이름과 실제파라메터이름이 같을수 있는가?

6.9 void함수에 return명령문이 반드시 있어야 하는가? 그 리유는 무엇인가?

6.10 void가 아닌 함수에 return명령문이 반드시 있어야 하는가 ? 그 리유는 무엇인가?

6.11 int형형식파라메터 n의 3제곱을 되돌리는 int형함수 Cube()를 작성하시오.

6.12 float형형식파라메터로서 3각형높이 h와 3각형의 밑변 w를 리용하여3각형의 면적을 계산하는 float형함수 Triangle()을 작성하시오.

6.13 float형형식파라메터로서 4각형의 높이 h와 4각형의 너비 w를 리용하여 4각형의 면적을 계산하는 float형함수 Rectangle()을 작성하시오.

6.14 표준출력흐름에 행바꾸기문자('\ n')를 2개 출력하는 void형함수 DoubleSpace()를 작성하시오. 함수에 의하여 현시되게 되는 행바꾸기문자의 개수를 나타내는 int형형식파라메터 n을 가지는 void형함수 EndLine()을 작성하시오.

6.15 사용자에게 반경을 입력할것을 요구하는 문자렬을 출력하고 사용자의 응답을 입력하여 되돌리는 float형함수 GetRadius()를 작성하시오.

6.16 프로그람 6-1을 련습문제 6.16에서 작성한 GetRadius()를 리용하여 다시 작성하시오.

6.17 2장에서 취급한것처럼 직선공식은 보통 y=mx+b이다. 방향결수 m과 y축단편 b, x자리표 x를 float형파라메터로 하는 함수 Line()을 작성하시오. 함수는 m과 b로 지정된 직선과 련관된 y자리표를 계산한다.

6.18 4개의 float형파라메터 m1, b1, m2, b2 을 가진 bool형함수를 작성하시오. 파라메터는 2개의 조로 되여 있다. 첫번째 조 m1과 b1은 첫번째 직선의 보조변수들이다.

6.19 두 직선이 평행이면 true를, 아니면 false를 돌린다. 함수에는 왜 4개의 파라메터가 필요한가?

6.20 아래의 기능을 수행하는 함수를 작성하시오. 어떤 값들이 파라메터이고, 국부상수인가 그것들의 형은 무엇인가를 결정하시오.

1) Speed(): 어떤 대상이 초기에 속도 v_0 m/s로 움직여 가속도 a m/s^2로 가속할 때 t초후의 대상의 속도를 계산한다. 공식은 속도=v_0+at이다.
2) Distance(): 초기속도가 0이고 가속도가 a인 어떤 대상이 t초동안 움직인 처리를 계산한다.
3) BarVolume(): 너비 w, 길이 l, 높이 h인 직6면체의 체적을 계산한다. 공식은 체적=wlh이다.
4) SphereVolume(): 반경이 r인 구의 체적을 계산한다. 공식은 체적=$4\pi r^3$ 이다.

6.21 다음의 프로그람의 출력결과는 무엇인가? 그 리유를 설명하시오.

```
#include <iostream>
#include <string>
using namespace std;
int i = 0;
int I = 1;
int main() {
    int i = 2;
    int I = 3;
    :: i = i + 10;
    I = ::I + I + 20;
    ::I = ::i + 30;
    i = I + 40;
    cout << i << endl;
    cout << ::i << endl;
    cout << I << endl;
    cout << ::I << endl;
    return 0;
}
```

6.22 다음의 프로그램의 출력결과는 무엇인가? 그 리유를 설명하시오.

```
#include <iostream>
#include <string>
using namespace std;
int counter = 0;
void f() {
    ++counter;
}
void g() {
    f();
    f();
}
void h() {
    f();
    g();
    f();
}
int main() {
    f();
    cout << counter << endl;
```

```
    g();
    cout << counter << endl;
    h();
    cout << counter << endl;
    return 0;
}
```

6.23 다음의 프로그램의 출력결과는 무엇인가? 그 리유를 설명하시오.

```
#include <iostream>
#include <string>
using namespace std;
void f() {
    int i = 1;
    int j = 2;
    cout << "f: i = " << i << endl;
    cout << "f: j = " << j << endl;
    return;
}
int main() {
    int i = 10;
    int j = 20;
    f();
    cout << "main: i = " << i << endl;
    cout << "main: j = " << j << endl;
    return 0;
}
```

6.24 다음의 프로그램의 출력결과는 무엇인가? 그 리유를 설명하시오.

```
#include <iostream>
#include <string>
using namespace std;
void f(int i, int j) {
    cout << "f: i = " << i << endl;
    cout << "f: j = " << j << endl;
    return;
}
int main() {
    int i = 10;
    int j = 20;
    f();
```

```
    cout << "main: i = " << i << endl;
    cout << "main: j = " << j << endl;
    return 0;
}
```

6.25 다음의 프로그람의 출력결과는 무엇인가? 그 리유를 설명하시오.

```
#include <iostream>
#include <string>
using namespace std;
void f(int i, int j) {
    i = i + j;
    j = j + i;
    cout << "f: i = " << i << endl;
    cout << "f: j = " << j << endl;
    return;
}
int main() {
    int i = 10;
    int j = 20;
    f(50, j);
    f(i, 50);
    cout << "main: i = " << i << endl;
    cout << "main: j = " << j << endl;
    return 0;
}
```

6.26 다음의 프로그람의 출력결과는 무엇인가? 그 리유를 설명하시오.

```
#include <iostream>
#include <string>
using namespace std;
void f(int i, int j) {
    int temp;
    temp = i;
    i = j;
    j = temp;
    cout << "f: i = " << i << endl;
    cout << "f: j = " << j << endl;
    return;
}
int main() {
```

```
    int i = 10;
    int j = 20;
    f(a, b);
    f(b, a);
    cout << "main: i = " << i << endl;
    cout << "main: j = " << j << endl;
    return 0;
}
```

6.27 4각형모양의 빵의 체적을 구하는 8개의 파라메터 w, l, h, m, b, n을 가진 SwissCheese()함수를 작성하시오. 여기서 w는 너비, l은 길이, h는 높이, m은 반경이 b인 구형식의 공기구멍들의 개수, n은 반경이 r이고 높이가 d인 원기둥형식의 구멍개수이다. 이 장에서 정의한 CylinderVolume()함수를 리용하여야 하며 련습문제 6.21에서 작성한 BarVolume()과 SphereVolume()을 리용하여야 한다. 그다음 빵의 특성값을 사용자에게서 입력하여 그 체적을 표시하는 프로그람을 완성하시오.

6.28 1니클은 5펜스이고 1다임은 10펜스이며 1쿠워터는 25펜스이고 0.5딸라는 50펜스이다. 다음의 함수들을 작성하시오. 매 함수는 하나의 int형형식파라메터 Amount를 가진다.

1) HalfDollars():Amount를 교환하는데 리용될수 있는 0.5딸라들의 최대수를 계산한다.
2) Quarters():Amount를 교환하는데 리용될수 있는 쿠워터들의 최대수를 계산한다.
3) Dimes():Amount를 교환하는데 리용될수 있는 다임들의 최대수를 계산한다.
4) Nickels():Amount를 교환하는데 리용될수 있는 니클들의 최대수를 계산한다.

6.29 **int**형파라메터 Amount를 가지며 우에서 본 화페단위들의 최소수를 리용하여 Amount를 교환하는 방법을 표준출력흐름 cout에 현시하는**void**형함수 MakeChange()를 작성하시오. 앞의 문제에서 작성한 함수들을 리용하여야 한다. 또한 사용자에게서 가격을 입력하여 MakeChange()함수를 호출하는 프로그람을 완성하시오.

6.30 함수 PromptAndGet()를 string서고에 있는 string형을 자료형으로 하는 파라메터 s를 가진 함수로 수정하시오. s는 입력재촉통보이다.

6.31 stock프로그람을 주당 세번째 자료값을 입력하는 프로그람으로 수정하시오. 추가적인 값은 주별 총화가격이다. 이 값을 파라메터로 리용하여 가격구간막대기의 적당한 위치에 표식을 붙이는 ChartWeek()함수를 작성하시오.

6.32 목록 6-1에 있는 W창문의 선언은 그의 크기가 자료를 분석하기전에 설정된다는 위치상의 문제가 있다. 자료가 파일에서 입력되므로 한번이상 정보를 처리할 가능성이 있다. 증권자료의 주소와 파일에서 가장 높은 증권가격을 결정하는 함수 NumberOfWeeks()와 HighStockPrice()를 작성하시오. stock.cpp에 있는 W의 정의를 아래와 같이 W의 초기화에서 이 함수를 리용하도록 고치시오.

```
extern const float ChartXOffset;
extern const float ChartYOffset;
string StockFile = GetFileName;
SimpleWindow W("Stock Char",
  HighStockPrice(StockFile) + (ChartXOffset+1),
```

NumberOfWeeks(StockFile) + (ChartYOffset+1);

6.33 Factorial()함수를 **double**형값을 되돌리도록 다시 작성하시오.

6.34 문제6.34에서 작성한 Factorial()함수를 리용하여 e의 값을 근사적으로 계산하는 프로그람을 작성하시오. e를 구하는 공식은 아래와 같다.

$$1+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+....$$

6.35 화씨온도를 섭씨온도로 바꾸는 **float**형함수 ftoc()를 작성하시오. 변환공식은 C=5(F-32)/9이다.

6.36 섭씨온도를 화씨온도로 바꾸는 **float**형함수 ftoc()를 작성하시오. 변환공식은 F=(9C/5)+32이다.

6.37 마일로 계산한 거리를 키로메터로 바꾸는 **float**형함수 mtok()를 작성하시오. 1mile=1.609344km

6.38 키로메터를 마일로 바꾸는 **float**형함수 ktom()을 작성하시오.

6.39 옹근수형파라메터i를 가지는 **bool**형함수 IsPrime()를 작성하시오. i가 씨수이면 **true**를, 아니면 **false**를 귀환한다.

6.40 **char**형파라메터 c를 가진 **bool**형함수 IsEndOfSentence()를 작성하시오. c가 끝점, 물음부호, 감탄부호이면 **true**를, 아니면 **false**를 귀환한다.

6.41 옹근수형파라메터를 가진 **int**형함수 Sum()를 작성하시오. 함수로부터 그 파라메터까지의 옹근수총합을 되돌려야 한다.

6.42 6.14에서 작성한 Factorial()함수를 그의 파라메터값이 부수이면 끝나도록 고치시오. 0!은 수학적으로 1이므로 수정한 함수는 이 계산에 영향을 미치지 않는다.

6.43 어떤 옹근수범위를 나타내는 두 옹근수를 입력으로 취하는 프로그람을 설계하시오. 프로그람은 이 범위내의 옹근수들의 합을 현시한다.

6.44 2개의 파라메터, 문자형 c와 옹근수형 n을 가진 **void**형함수 PrintChar()를 작성하시오. 함수는 표준출력흐름 cout에 n개의 c를 출력하여야 한다.

6.45 4개의 파라메터 a, b, c, x 를 가진 **float**형함수 EvaluateQuadraticPolynomial()을 작성하시오. 함수는

$$ax^2 +bx+c$$

의 값을 돌려야 한다.

6.46 $a_3x^3 +a_2x^2 +a_1 x+a_0$ 곡선과 x축, x=b, x=t로 둘러 막힌 구역의 면적을 구하는 double형함수 CubicArea()를 작성하시오.

6.47 방정식 $a_4x^4 +a_3x^3 +a_2x^2 +a_1 x+a_0$ 의 곡선제형면적을 구하시오. 곡면의 x축구간은 (s, t)이다. **double**형함수 QuadticArea()를 작성하시오.

6.48 각은 보통 °나 라디안으로 측정한다. 여기서 360 °는 2π라디안과 같다. **float**형파라메터를 가진 **float**형함수 DegreesToRadians()는 °를 라디안으로, RadiansToDegrees()는 라디안을 °C로 바꾸는 함수이다.

6.49 현재입력행을 입력하여 처리하는 StripVowels()함수를 작성하시오. 함수는 입력렬에서 자음만을 문자렬로 되돌린다.

6.50 km로 표시한 거리를 광년으로 계산하는 함수 DistanceToLightYears()를 작성하시오. 빛의 속도는 대략 2.997925×10^8m/s이다. 파라메터와 되돌림값의 자료형은 무엇이여야 하는가?

6.51 string형파라메터 W를 가진 string형함수PigLatin()을 작성하시오. 함수는 w문자렬의 매 단어들

을 PigLatin형식으로 바꾼 문자렬을 되돌린다. 콤파일러규칙은 련습문제 5.41에 있다.

6.52 두개의 double형파라메터 t와 w를 가진 double형함수 WindChill()을 작성하시오. 여기서 t는 온도이고 w는 바람속도이다. 함수의 되돌림값은 련습문제 5.42에 있는 공식에 따라 계산되는 온도이다.

6.53 두개의 파라메터 t와 h를 가진 double형함수 Index()를 작성하시오. t는 온도이고 d는 습도이다. 함수의 되돌림값은 련습문제 5.43에 있는 공식에 따라 계산되는 온도이다.

6.54 다음의 함수를 분석하시오.

```
void scramble(int i, int &j, int k) {
    i = 10;
    j = 20;
    k = 30;
    return;
}
```

1) 다음의 프로그람의 출력은 무엇인가?

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    int i = 1;
    int j = 2;
    int k = 3;
    scramble(i, j, k);
    cout << "i =" << i << "j =" << j << "k =" << k << endl;
    return 0;
}
```

2) 다음의 프로그람의 출력은 무엇인가?

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    int i = 1;
    int j = 2;
    int k = 3;
    scramble(j, j, j);
    cout << "i = " << i << "j = " << j << "k = " << k << endl;
    return 0;
}
```

6.55 다음의 함수를 분석하시오.

```
void scramble (int i, int &j, int &k) {
    i = 10;
    j = 20;
    k = 30;
    return;
}
```

1) 다음의 프로그램의 출력은 무엇인가?

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    int i = 1;
    int j = 2;
    int k = 3;
    scramble(k, j, i);
    cout << "i = " << i << "j = " << j << "k = " << k << endl;
    return 0;
}
```

2) 다음의 프로그램의 출력은 무엇인가?

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    int i = 1;
    int j = 2;
    int k = 3;
    scramble( j , j,  j );
    cout << "i = " << i << "j = " << j << "k = " << k << endl;
    return 0;
}
```

6.56 다음의 함수를 분석하시오.

```
void scramble (int i, int &j, int &k) {
    i = 10;
    j = 20;
    k = 30;
    return;
```

```
}
```

1) 다음의 프로그램의 출력은 무엇인가?

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    int i = 1;
    int j = 2;
    int k = 3;
    scramble(k, j, i);
    cout << "i = " << i << "j = " << j << "k = " << k << endl;
    return 0;
}
```

2) 다음의 프로그램의 출력은 무엇인가?

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    int i = 1;
    int j = 2;
    int k = 3;
    scramble(j, j, j);
    cout << "i = " << i << "j = " << j << "k = " << k << endl;
    return 0;
}
```

6.57 다음의 C++프로그램을 분석하시오. 출력결과는 어떠한가?

```
#include <iostream>
#include <string>
using namespace std;
int funny(int &a, int b) {
    int C = a + b;
    a = b;
    b = c;
    return C;
}
int main() {
    int x = 3;
```

```
    int y = 4;
    int z = 5;
    cout << funny(x, y) << endl;
    cout << "X is: " << x << "and y is: " << y << endl;
    z = funny(x, z);
    cout << "z is: " << z << " and x is: " << x << endl;
    return 0;
}
```

6.58 다음의 프로그람을 분석하시오.

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    int x = 1;
    f(x);
    cout << "x is " << x << endl;
    return 0;
}
```

이 프로그람이 실행할 때 출력은

```
x is 2
```

이다. void형함수 f( )의 원형을 작성하시오.

6.59 다음의 함수 tricky()를 분석하시오.

```
void tricky(const int x, int &y, int z) {
    x = 3;
    y = 3;
    z = 3;
}
```

함수가 콤파일될 때 어느 값주기명령문에서 오유가 생기는가? 콤파일오유가 생기면 값주기명령문 옆에 Yes를 쓰고 아니면 No를 쓰시오.

1) x=3; ____________
2) y=3; ____________
3) z=3; ____________

6.60 다음의 C++프로그람의 출력은 무엇인가?

```
#include <iostream>
#include <string>
using namespace std;
```

```
void f(int &a) {
    cout << "in" << a << "\n";
    return;
}
void f(char &a) {
    cout << "char" << a << "\n";
    return;
}
int main() {
    int i = 1;
    char c = ' C';
    f(i);
    f(c);
    return 0;
}
```

6.61 다음의 프로그램에서 오류를 찾으시오.

1)
```
int Update(const int &x, int y, int z) {
    y = z % 3;
    x = y + z;
    return x;
}
```

2)
```
int Sum(int x, int y = 3, int z) {
    return x + y + z;
}
```

3)
```
void Flow(double x, int y) {
    return x + y;
}
```

4)
```
double Mul(const double a, const double b) {
    return a * b;
}
```

5)
```
double Mul(double x, double y) {
    return x * y;
}
```

6)
```
#include <iostream>
#include <string>
using namespace std;
bool GetInput(istream in, int &Value) {
    if(in >> Value)
```

```
            return true;
        else
            return false;
    }
```
7)
```
#include <iostream>
#include <string>
using namespace std;
bool GetInput(istream &in, int &Value) {
    if(in >> Value)
        return true;
    else
        return false;
}
```

6.62 number.dat라는 파일에서 전화번호를 읽어 들이는 프로그람을 작성하시오. 프로그람은 입력값이 유효한 전화번호인가를 결정하여야 한다. 유효한 전화번호형식은

D D D '-' D D D '-' D D D D

이다. 여기서 첫번째 D는 령이 아니다. 전화번호가 맞으면 전화번호를 출력한다. 그다음 i를 표시하고 《옳음》이라고 표시한다. 전화번호가 틀리면 《틀림》이라고 표시한다.

6.63 두개의 문자를 입력하기 위한 재촉통보를 내보내는 프로그람을 작성하시오. 프로그람은 두개 문자가 유효한 략자인가를 결정한다. 략자가 옳으면 프로그람은 그 두 문자뒤에 :를 찍고 《옳음》이라고 표시하며 아니면 《틀림》이라고 표시한다. 실례로 입력값이

Va

라면 프로그람은 다음과 같은 문자를 현시한다.

VA : 옳음

입력값이

tz

라면 프로그람은 다음과 같은 문자를 현시한다.

TZ : 틀림

6.64 다음의 함수를 분석하시오.

```
#include <iostream>
#include <string>
using namespace std;
void f(int a, double b) {
cout << "f(int, double) says a is " << a << endl;
cout << "f(int, double) says b is " << b << endl;
return;
```

```
}
void f(int a, int b) {
cout << "f(int, int) says a is" << a << endl;
cout << "f(int, int) says b is" << b << endl;
return;
}
void f(double a, double b) {
cout << "f(double, double) says a is " << a << endl;
cout << "f(double, double) says b is " << b << endl;
return;
}
```

1) 다음의 프로그램이 실행될 때 출력결과는 무엇인가?

```
int main() {
    int i = 1;
    int j = 2;
    double x = 3.5;
    double y = 10.2;
    f(i, j);
    f(i, y);
    f(y, x);
    return 0;
}
```

2) 다음의 프로그램이 실행될 때 출력결과는 무엇인가?

```
int main() {
    f(1, 2.3) ;
    f(2, 4);
    f(2.6, 10.5);
    return 0;
}
```

3) 프로그램이 콤파일될 때 콤파일오유가 생긴다. 무엇이 잘못인가?

```
int main() {
    f(1, 2.3) ;
    f(2.3, 4);
    f(2.6, 10.5);
    return 0;
}
```

6.65 프로그램 6-9에 있는 Valid()에는 아래와 같은 if명령문이 있다.

```
if (d4 == 0)
        return false;
    else
        return ((d1 + d2 + d3) % d4) == d5;
```

하나의 **return**명령문만을 가지도록 **if**명령문을 다시 작성하시오. 어느 형식이 명백한가?

6.66 다음의 프로그램의 출력결과는 무엇인가?

```
#include <iostream>
#include <string>
#include <assert.h>
using namespace std;
void DisplayString(ostream &sout, char c = '*',
  int count = 1) {
    assert(count >= 0);
    for (int i = 0; i < count; ++i)
    sout << c;
    sout << endl;
}
int main() {
    DisplayString(cout);
    DisplayString(cout, '-', 10 );
    DisplayString(cout, '+');
    return 0;
}
```

6.67 호출코드를 파일로부터 읽어 들이도록 프로그램 6-9를 수정하시오. 호출코드는 access.dat라는 파일에 있는데 한행당 한개 코드가 대응된다. 매 호출코드에 대하여 코드가 맞는가 맞지 않는가를 결정한다. 매 코드뒤에 《옳음》,《틀림》이라는 단어를 붙여서 tested.dat라는 파일에 쓰기한다.

6.68 한개 학년의 평균성적을 계산하는 프로그램을 작성하시오. Scores.dat라는 파일에서 자료를 입력한다. Scores.dat는 머리부행과 매 학생들의 성적으로 구성되였다. 머리부행의 형식은 다음과 같다.

n 　$무게_1$ $무게_2$…$무게_n$

여기서 n은 매 학생의 성적들의 수이고 $무게_i$는 i번째 성적의 무게이다. 머리부행다음에 매 학생의 성적이 있다. 형식은

이름 　$성적_1$ $성적_2$…$성적_n$

이다. 여기서 이름은 학생의 이름이고 $성적_i$는 i번째 과목의 성적이다. 성적은 0부터 100사이의 값이다. score.dat라는 파일을 읽어 average.dat라는 파일에 다음의 형식으로 출력한다.

이름 　$성적_1$ $성적_2$…$성적_n$ => nn.nn평균

여기서 nn.nn은 그 학생의 성적들에 대한 무게 붙은 평균이다. 무게 붙은 평균은

$$\frac{\sum_{i=1}^{n} score_i \times weight_i}{n}$$

로 계산한다. 프로그람에서 유효성검사를 진행하여야 한다. 즉 매개 성적이 0~100사이에 있는가를 확인하고 매 학생이 n과목성적을 다 가지고 있는가를 확인하여야 한다. 학생의 성적이 틀린다면 프로그람은 cerr에 오유통보를 내고 출력파일 average.bat에 쓰기동작을 하지 말아야 한다. 풀이방향: 프로그람에는 입력값을 처리하는 보조프로그람과 유효성검사를 위한 보조프로그람이 있어야 한다.

6.69 문제 6.68을 확장하여 등수를 결정하는 프로그람을 작성하시오. 등수기준은 아래와 같다.

평균성적	**등수**
0-59	5
60-69	6
70-79	7
80-89	8
90-100	9

average.dat파일에는 아래의 형식으로 출력한다.

이름 성적$_1$ 성적$_2$…성적$_n$ => nn.nn평균(등수)

6.70 문제 6.69를 확장하여 학년전체의 평균성적을 계산하여 출력하는 프로그람을 작성하시오.

6.71 흥미 있는 야구통계는 타률이다. 이 통계값은 뽈을 치는 능력을 측정한다. 타률은 타석에 들어 선 총수를 뽈을 쳐낸 회수로 나눈 값이다. 실례로 12개 타석에서(희생타는 타석에 포함하지 않는다.) 한 타수가 1루 2개, 2루 4개, 홈란 4개, 스트라이크아웃 0개라면 이 타수의 타률은 10÷12=0.833이다. 타률 0.690의 기록을 보유한 타수도 있다. 일련의 타수들의 타률을 계산하는 프로그람을 작성하시오. 타수들의 자료는 파일에 포함되여 있다. 파일은 다음과 같은 형식을 가지고 있다.

이름 타석 1루 2루 3루 홈란 스트라이크아웃

여기서 이름은 타수의 성이다. 다른 항목들은 옹근수값이다. 프로그람은 자료를 포함하는 파일의 이름입력을 사용자에게 독촉한다. 프로그람은 slugging.dat라는 새로운 파일을 창조한다. 이 파일은 타률을 계산한 값을 표현하고 있다. 이 파일의 매행은 다음과 같은 형식으로 서술되여 있다.

이름 쳐낸 수:bb 타석수:aa 타률:.nnn

이름은 타수의 성이고 bb는 뽈을 친 총 개수이며 aa는 타석들의 개수이고 .nnn은 타률이다.

6.72 시간을 표시하는 인수를 가진 TimeToSeconds()함수를 작성하시오. 시간을 표현하는 형식은 다음과 같다.

hh:mm:ss

여기서 hh는 시간이며 mm은 분, ss는 초이다. TimeToSeconds()함수는 초값을 되돌린다. TimeToSeconds()함수가 무효한 시간(실례로 02:23:67)을 받으면 assert()를 리용하여 오유를 내보낸다. 시간을 나타내는 문자렬을 입력하는 프로그람작성에서 이 함수를 시험해 보시오. 이때 프로그람은 입력시간을 표시하고 초를 표시한다. 다음의 시간을 가지고 함수를 시험해 보시오.

1) 00:00:00	7) 10:50:05
2) 00:00:59	8) 00:59:49
3) 00:00:60	9) 03:20:20
4) d00:01:30	10) 05:55:30
5) 00:44:30	11) 30:00:00
6) 00:60:59	12) 11:11:11

6.73 두 파라메터 EndTime과 StartTime을 가진 ElapseTime()함수를 작성하시오. EndTime과 StartTime은 다음의 형식으로 시간을 나타 내는 문자렬이다.

hh:mm:ss

함수는 StartTime으로부터 EndTime까지의 초수를 계산한다. 함수에서 StartTime은 사용자가 설정할수 있는 파라메터이다. StartTime이 주어 지지 않으면 기정적으로 00:00:00이다. 함수에서는 반드시 6.73번 문제에서 작성한 TimeToSecond()함수를 리용해야 한다. StartTime과 EndTime이 무효하거나 StartTime이 EndTime보다 크면 오유를 내야 한다. 시험프로그람을 작성하시오. 시험자료는 아래와 같다.

시작시간	끝시간
00:00:00	00:01:00
00:20:00	00:25:50
01:30:40	00:35:50
01:10:51	01:11:61
20:10:10	21:09:08

6.74 시간을 나타내는 파라메터를 가진 FormatTime()함수를 작성하시오. 함수는 우의 문제와 같은 형식으로 시간을 나타내는 문자렬을 되돌린다.

hh:mm:ss

시험프로그람을 작성하시오. 시험자료는 아래와 같다.

시간
0
59
60
3600
10000
50000

6.75 두 옹근수의 최대공통약수를 계산하는 재귀함수를 작성하시오. 두 옹근수의 최대공통약수는 그 두 수를 다 나눌수 있는 가장 큰 옹근수이다. 최대공통약수들의 공식은

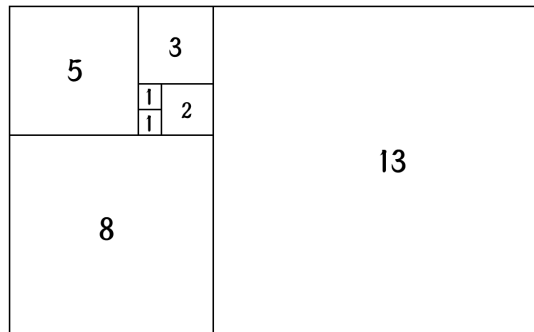
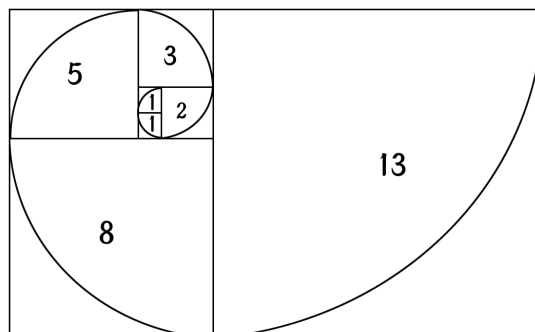
$$gcd(m,n)=\begin{cases} n & \text{n을 m으로 나누는 경우} \\ gcd(\text{m을 n으로 나눈 나머지}) & \text{그밖의 경우} \end{cases}$$

6.76 사용자로부터 수를 입력하며 다음식의 값이 입력한 수 n과 가장 근사하게 되는 i를 계산하고 표시하는 프로그람을 작성하시오.

$$\frac{1}{\sqrt{5}}\cdot\left(\left(\frac{1+\sqrt{5}}{2}\right)^{i}-\left(\frac{1-\sqrt{5}}{2}\right)^{i}\right)$$

피보나치수렬과 결과가 같은가 비교하시오.

6.77 피보나치수렬에서 매수들사이의 비률을 따지면 어떤 값으로 다가간다는것을 알수 있다. 례하면 5/3는 1.66666, 8/5은 1.6, 13/8은 1.625이다. 이런 비률을 《황금비률》 혹은 《황금비례중항》이라고 한다. 황금비률의 근사값을 계산하는 프로그람을 작성하시오. 정확도평가를 하자면 피보나치수렬의 황금비률을 계산한다. 계산은 황금비률의 차가 0.0005이하일 때까지 진행된다.

6.78 아래의 그림은 피보나치수렬의 관계를 보여 주는 그림이다.

이 그림은 변의 길이가 1인 2개의 바른4각형으로부터 시작하여 그렸다. 오른쪽에 변의 길이가 그전 2개의 바른4각형의 변길이의 합인 바른4각형이 그려 진다(즉 변의 길이가 2인 바른4각형). 매개 4각형의 변의 길이는 마지막으로 그린 2개의 4각형의 변의 길이의 합이다. 이런 방식으로 그린 도형 즉 피보나치 4각형을 만들자. 피보나치 4각형을 그리는 RectangleShape를 리용하는 Ezwindow프로그람을 개발하시오. 프로그람은 바른4각형의 개수를 입력하기 위한 입력재촉문이 표시되여야 한다. 매 4각형마다 4등분원을 그리면 아래와 같은 라선이 얻어 진다. 이 라선은 조가비나 달팽이조가비와 같이 자연계에서 있는 라선과 비슷하다.

제7장. 클라스구조와 객체지향설계

소 개

객체는 C++와 같은 객체지향언어에서 프로그람작성의 기본단위이다. 객체는 정보의 모형이며 속성과 동작을 둘다 교갑화할수 있는 쏘프트웨어묶음으로 실현된다. C++에서 교갑화된 객체들의 새로운 형을 정의하기 위한 구조가 바로 클라스이다. 클라스는 객체를 만들어 내는 《설계도》이다. 이 장에서는 객체들의 명세부인 이러한 《설계도》를 만드는 방법을 설명한다. 여기서는 이전의 장에서 정의한 프로그람작성자정의형인 RectangleShape 클라스선언을 통하여 그것을 보여 준다.

기본개념

- 클라스구조
- 정보은페
- 교갑화
- 자료성원
- 성원함수
- 구축자
- 검토자
- 변이자
- 촉진자
- const 함수
- public 와 private 속성을 가진 객체의 접근지정
- 객체지향분석과 설계

7.1 프로그람작성자정의자료형

앞장에서는 프로그람작성의 기초와 쏘프트웨어공학의 련습을 주었다. 그로부터 얻은 지식이면 간단한 문제들은 자체로 풀수 있다. 그러나 좀 더 복잡한 문제를 해결하기 위해서는 프로그람언어의 보충적인 특성, 알고리듬작성방법과 공학기술에 정통해야 한다. 이 장에서는 새로운 객체의 형들을 정의하기 위한 C++의 클라스구조에 대하여 소개한다. 이 구조는 C++를 리용하는 객체지향프로그람작성에서 기초로 된다.

기본형(foundamental-type)객체는 이 프로그람언어에 의해 제공되는 객체이다. 실례를 들어 C++에서 int형과 그 연산자는 C++의 언어정의부분이다. 즉 모든 C++콤파일러에서는 int객체를 제공해야 한다. 기본형들외에 C++는 다른 형을 정의하기 위하여 여러가지 기구를 제공한다. 그 다른 형들을 프로그람작성자정의형 혹은 사용자정의형이라고 부른다.

앞장에서 새로운 형을 창조하기 위하여 C++ enum구조를 사용하였다. color형은 RectangleShape 객체와 함께 리용되는데 그 참조에 이러한 enum구조가 리용되였다. C++에서는 enum구조외에 새로운 형을 창조하기 위한 여러가지 다른 메쏘드 (method) 들도 제공된다. 실례로 프로그람작성자는 C++에 의해 객체들의 목록형을 정의할수 있다. 이 객체를 배렬(array)이라고 부른다. 다른 례는 프로그람작성자가 그 값들이 다른 객체의 기억주소로 되는 형을 정의할수 있게 하는 지적자기구이다. 이 2개의 기구들은 다음장들에서 참고하도록 한다

이 장에서 소개되는 클라스구조는 새로운 형을 정의하기 위한 가장 중요한 기구이다. 이 구조에 의하여 쏘프트웨어기술자들은 속성요소과 행동요소을 둘 다 포함하거나 교갑화하는 클라스형객체를 정의

할수 있다.

클라스구조에 의하여 새로운 객체의 형을 창조하는 방법을 설명하기 위하여 앞선 장들에서 리용한 프로그람작성자정의형의 선언 즉 RectangleShape에 대하여 고찰한다. RectangleShape설계에서 목표는 창조하기도 쉽고 사용하기도 쉬운 도형처리객체를 얻는것이다. RectangleShape가 실지의 객체를 묘사할수 있다면 앞의 장들에서 본것처럼 사용할수 있다. 클라스형을 창조하는데서 처음단계는 객체의 속성을 결정하는것이다. 그라프현시체계에서 객체를 위한 4각형의 필요한 속성은 그 크기와 보기구역이나 현시창문, 그것이 현시되는 창문에서의 위치와 색이다. C++에서 클라스형객체의 속성들은 클라스의 자료성원으로 간주된다. 이 장의 마지막에서 클라스들이 어떻게 정의되는가를 고찰할 때 자료성원들이 단지 클라스형객체의 부분객체라는것을 알게 된다.

클라스형객체를 창조하는 두번째 단계는 객체가 접수할수 있는 통보문들과 객체상에서 수행할수 있는 연산을 정의하는것이다. 이것은 그의 행동요소이다. 클라스형객체의 행동요소는 어떠한 기능을 수행하도록 객체에로 통보문을 보낼수 있는 성원함수들의 모임이다. 창문화된 도형현시체계에서 RectangleShape에 맞는 클라스를 계속 개발하려면 4각형은 2개의 통보문클라스를 조종할것을 요구하게 된다. 한 통보문모임은 4각형의 속성값을 돌려 주며 다른 모임은 그의 속성값을 변경시킨다.

속성값을 돌려 주는 통보문들을 검토자(inspector)라고 한다. 최대의 유연성을 보장하기 위하여 RectangleShape의 정의에 모든 자료성원들에 대한 검토자들이 포함되게 된다. 이리하여 RectangleShape검토자들은 GetColor(), GetSize(), GetWidth(), GetHeight(), GetPosition(), GetWindow()를 가져야 한다. 검토자가 일부 기능의 복사를 제공한다는데 주의하여야 한다. 4각형의 길이와 높이를 돌려 주는 GetSize()외에 GetWidth(), GetHeight()가 있다. 이 검토자들은 의뢰기가 길이 혹은 높이를 한꺼번에가 아니라 따로따로 꺼내려고 하는 경우에 편리하게 쓰인다.

속성을 변경하거나 설정하는 통보문들을 변이자(mutator)라고 한다. 다시 말하여 RectangleShape를 될수록 유연하게 만들기 위하여 그의 위치와 출현을 조종하는 속성들에 대한 변이자를 가져야 한다. 이러한 RectangleShape의 변이자는 SetColor(), SetPosition(), SetSize()이다. 매 성원함수들은 이름으로 지적되는 속성을 설정한다.

검토자도 아니고 변이자도 아닌 통보문이 하나 더 있다. 그리기통보문은 4각형을 창문에 현시하라는것을 지적한다. Draw()는 촉진자의 한가지 실례이다. 촉진자(facilitator)는 객체가 일부 동작이나 봉사를 수행하도록 한다.

RectangleShape에 검토자 GetWindow()는 있지만 그에 대응한 변이자 SetWindow()가 없다. RectangleShape가 일단 창조되면 그것은 현시창문과 결합되며 현시창문으로 결합만 되면 그것은 변경할수 없다. 그러므로 필요한 변이자를 제공하지 않는 방법으로 사용자가 창문을 바꾸는것을 막을수 있다.

그림 7-1은 간단한 도형현시체계에서의 4각형에 대한 추상화를 소개한다. 점으로 찍은 구름도형은 이 구체례가 4각형의 클라스정의이며 실제적인 4각형의 구체례는 아니라는것을 말하여 준다. 편의상 문자 C로 클라스를, 줄임말 DM으로 자료성원을, MF로 성원함수를 현시한다.

왜 클라스 RectangleShape가 4각형의 추상화라고 하는지 의문이 생길수도 있다. 후에 크기, 색갈, 창문에서의 위치를 가지고 4각형에 대해서 좀 더 자세히 설명한다. 추상화는 적합치 않은것들은 소거하고 객체의 본질적인 특성들에 초점을 둔다. 우리의 경우 그 본질적인 속성들이란 도형처리프로그람들을 개발하는데 쓸모가 있는 RectangleShape를 만드는데 요구되는것들로서 창문안에서의 4각형의 크기, 색갈, 위치이다. 그리고 적합치 않은것이란 RectangleShape가 창문에 어떻게 그려 지는가 하는것이다. 4각형의 추상화에서는 그에 대해서 전혀 언급하지 않는다. 비본질적인 세부들을 제거하면

RectangleShape를 사용하기가 쉽다. 더우기 RectangleShape를 어떻게 리용하는가 하는것은 리용하는 창문체계에 관계되지 않는다. 이렇게 프로그람들은 수정함이 없이 여러 조작체계상에서 실행될수 있다.

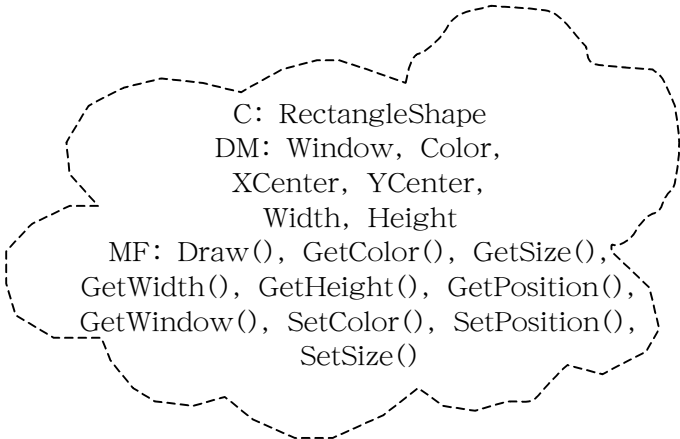



그림 7-1. RectangleShape 클라스의 자료성원과 성원함수

7.2 RectangleShape 클라스

지금까지 4각형에 대한 추상화를 결정함으로써 객체추상화에 해당한 C++클라스를 어떻게 선언하는가 하는것을 고찰할수 있게 되였다. 클라스형객체선언에 대한 문법은 다음과 같다.

선언은 예약어 class(그뒤에 클라스이름이 놓인다.)로부터 시작된다. 클라스선언의 내부에는 검토자, 변이자 그리고 촉진자를 설정하는 함수의 원형이 있다. 클라스선언은 또한 클라스의 한 부분인 임의의 자료성원에 대한 정의도 포함하고 있다. 례를 들어 RectangleShape에 대한 클라스선언은 다음과 같다.

```
class RectangleShape {
    public:
        RectangleShape(SimpleWindow &Window,
        float Xcoord, float Ycoord, const color &Color,
        float Width, float Height);
        void Draw();
        color GetColor() const;
        void GetSize(float &Width,  float &Height) const;
        float GetWidth()  const;
        float GetHeight()  const;
        void GetPosition(float &Xcoord;
        float &Ycoord)  const;
        SimpleWindow& GetWindow() const;
        void SetColor(const color &Color);
        void SetPosition(float  Xcoord, float Ycoord);
        void SetSize(float Width, float Height);
    private:
        SimpleWindow &Window;
        float XCenter;
        float YCenter;
        color color;
        float Width;
        float Height;
};
```

다음의 프로그람은 창문에서 2×3cm 의 크기를 가진 R라는 푸른색 4 각형을 창조하기 위하여 우의 클라스를 리용한다.

```
#include "rect.h"
SimpleWindow W("MAIN WINDOW", 8.0, 9.0);
int ApiMain() {
    M.Open();
    RectangleShape R(W, 4.0, 4.0, blue, 2.0,3.0);
    R.Draw();
    return 0;
}
```

그림 7-2 는 결과창문을 보여 주고 있다.

C++클라스선언은 클라스의 이름이 뒤따르는 열쇠단어 class로부터 시작된다. 예상할수 있는것처럼 클라스의 이름은 유효한 C++식별자(identifier)이여야 한다. 클라스의 자료성원들과 성원함수들은 대괄호로 현시해 준다. 이 대괄호안의 자료성원들과 성원함수들은 **public**와 **private**로 현시된 두 부분으로

나누어 진다. 지금은 이 표식들의 의미를 론의하지 않는다.

예약어 **private**뒤에 오는 자료속성선언을 고찰하자. 하나의 선언은 객체의 매 속성에 대응하고 있다. 례를 들어 선언

```
SimpleWindow &Window ;
```

는 RectangleShape객체를 현시하게 될 SimpleWindow의 참조를 보유하는 Window라는 부분객체를 선언한다.

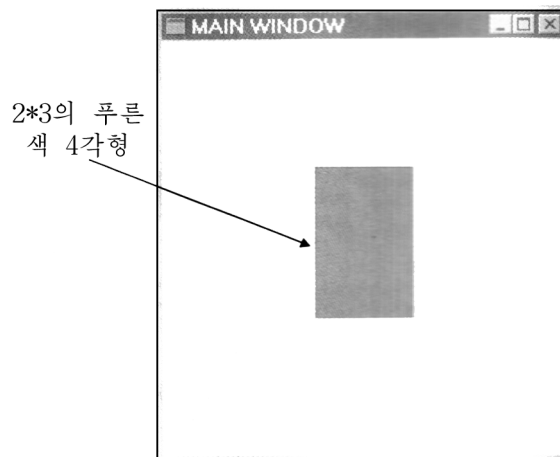



그림 7-2. SimpleWindow W에서의 RectangleShape R

왜 선언이 참조연산자 &를 리용하는가가 의문시될수 있다. 만일 선언을

```
SimpleWindow Window;
```

로 하였다면 RectangleShape는 하나의 SimpleWindow객체를 포함한다(객체에 대한 참조는 아니다). 값파라메터에 의한 호출에서와 같이 이 선언은 하나의 새로운 RectangleShape객체가 구체례화되면 하나의 Simplewindow객체의 복사가 이루어 진다는것을 의미한다. 이 기능은 하나의 SimpleWindow객체의 복사를 할 때 어떻게 되겠는가가 명백하지 않으므로 혼란을 가져오게 한다. 표준적으로 하나의 객체를 복사할 때 다른 하나의 꼭 같은 객체를 얻게 된다. RectangleShape를 창조할 때마다 다른 창문이 나타나게 하지 않으려면 SimpleWindow객체를 복사하기보다는 그에 대한 참조를 가지는것이 더 낫다. 우리는 참조파라메터넘기기기구에서와 같이 참조속성 Window를 그 객체를 포함하는 실제창문에 대한 별명으로 생각할수 있다.

선언

```
float XCenter;
float YCenter;
```

은 4각형의 중심자리표를 지적하는 부분객체들에 대한 선언이며 선언

```
color Color;
```

는 4각형의 색갈을 지적하는 자료성원을 선언하고 있다. 4각형의 크기와 관련한 속성들은 다음과 같은 자료성원을 가지게 된다.

```
float  Width;
```

```
float Height;
```

지금까지 리해한 RectangleShape의 속성들에 기초하여 여러가지 공개부함수들을 고찰해 보자. RectangleShape클라스선언에서 첫 원형

```
RectangleShape(SimpleWindow &Window,
float Xcoord, float Ycoord,  const color &Color,
float Width, float Height);
```

은 RectangleShape에 대한 구축자이다. 구축자함수는 객체가 정의되면 자동적으로 호출된다. 그 구축자는 객체의 자료성원들의 일부 혹은 전체를 적당한 값으로 초기화할수 있다. 구축자함수는 클라스와 같은 이름을 가지기때문에 쉽게 알수 있다. 정의

```
RectangleShape R(W, 4.0, 4.0, Blue, 2.0, 3.0);
```

은 R라고 하는 RectangleShape를 창조하고 구축자를 호출하며 R의 자료성원들을 초기화하기 위하여 거기에 값 W, 4.0, 4.0, Blue, 2.0, 3.0을 할당한다. 추측할수 있는것처럼 Window는 값 W로 초기화되며 XCenter와 YCenter는 값 4.0, 4.0으로, Color는 Blue로, Width와 Height는 2.0과 3.0으로 각각 설정된다. 객체를 창조하고 그의 자료성원들을 초기화하는 처리를 구체례만들기(instantiation)라고 한다.

마찬가지로 정의

```
RectangleShape R1(W, 1.0, 4.0, Cyan, 3.0, 3.0);
RectangleShape R2(W, 6.0, 4.0, Red, 1.0, 2.0);
```

은 R1과 R2라는 RectangleShape형의 두 객체들을 구체례로 만든다. 객체를 그 추상화로부터 특수한 구체례로 만드는 처리를 그림 7-3에 보여 주었다.

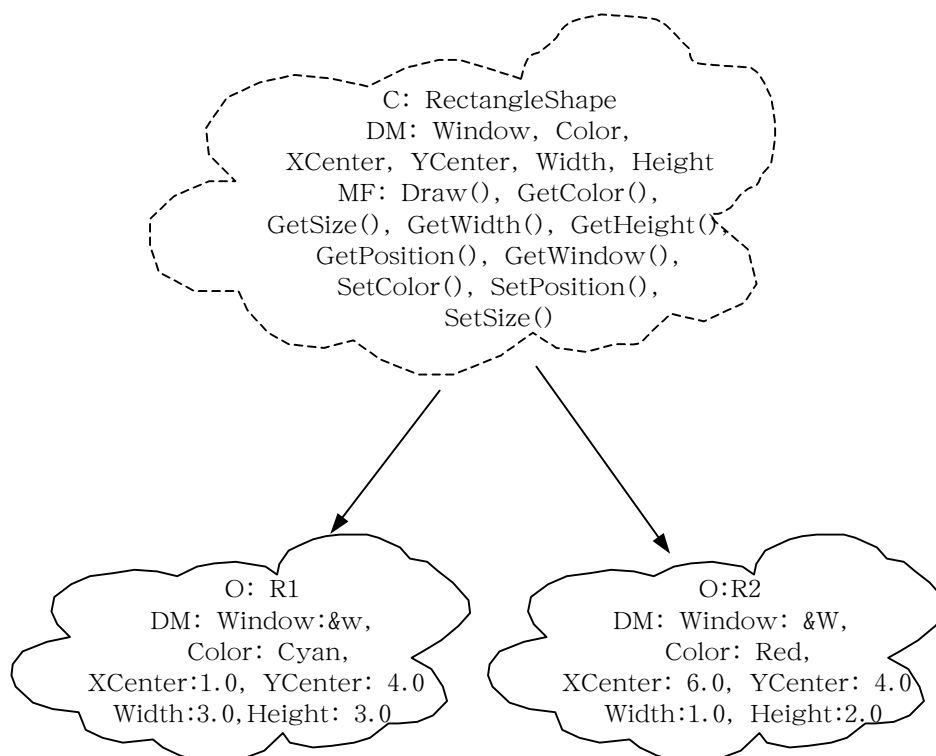
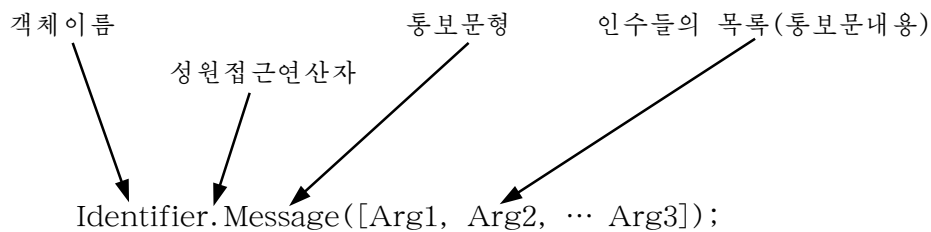



그림 7-3. RectangleShape객체 R1 과 R2의 구체례만들기

실선으로 된 구름도형은 그것이 추상화의 실제적인 구체례라는것을 의미한다. 편리상 객체의 이름을 문자 O로 현시한다. 자료성원들과 성원함수들은 클라스의 설명에 리용되였던 기호법으로 현시된다. RectangleShape정의의 다음부분에서는 RectangleShape객체가 접수할수 있는 통보문의 형을 렬거한다.이 함수들과 구축자들의 정의는 다른 모듈에 포함된다. 비록 클라스선언에서 성원함수들과 구축자의 정의를 포함하는것이 가능하다 해도 정보은폐의 원칙에 따라 개별적인 모듈에서 구체적으로 서술하도록 한다. 구축자나 성원함수들의 실현부를 클라스내에 포함시키면 클라스선언이 복잡해 지며 클라스에 대한 대면부가 더 보기 어려워 진다.

C++에서 통보문들은 성원함수를 통하여 객체에 보내여 진다. 성원함수호출에 의하여 객체에 통보문들을 보내기 위한 문법은 다음과 같다.

식별자와 통보문형사이의 점을 성원접근연산자(member access operator)라고 한다. 성원접근연산자는 객체의 성원함수나 자료성원을 선택한다.

클라스 RectangleShape는 8개의 성원함수를 포함한다. 첫번째 성원함수 Draw()는 객체가 창문에 자기자체를 나타내도록 하라는 통보문을 보낸다. RectangleShape가 구체례화되면 그것은 창문에 나타나지 않는다. 다시 말하여 프로그람작성자는 반드시 객체그리기통보문을 보내여 창문에 그리기 위한 RectangleShape객체를 명백하게 서술하여야 한다. 례하면 RectangleShape객체 R에 대하여 코드

```
R.Draw();
```

는 SimpleWindow에 R를 그리라는 통보문을 보낸다.

그리기함수를 왜 쓰는가 하는 의문이 제기될수 있다. 왜 객체가 구체례화될 때 창문에 나타나지 않는가? 이것은 있을수 있는 의문이다. 보게 되겠지만 명백한 그리기함수를 가지면 RectangleShape를 더 유연하게 할수 있다. 사람들은 흔히 도형의 성질들을 정확히 알기도전에 그 도형을 창조하려고 한다. 류사하게 새로운 색갈을 주고 위치를 변화시키며 새로운 위치에 그리기하는것으로 현재의 도형을 재리용하려고 한다. 그리기의 기능으로부터 구체례화기능을 떼내여 기정값으로 어떠한 형태를 창조하고 그의 속성들을 결정하여 설정하며 그다음 준비가 되면 그리기통보문을 보내여 도형이 나타나도록 할수 있다.

그리기성원함수선언뒤에 성원함수 GetColor(), GetSize(), GetWidth(), GetHeight(), GetPosition(), GetWindow()에 대한 선언들이 놓인다. 이것들은 RectangleShape의 속성을 돌려 주는 검토자들에 대한 선언이다. 검토자들인 GetSize()와 GetPosition()들이 둘 다 속성값들을 돌려 주는 참조파라메터들을 리용한다는데 주의하시오. 례를 들어 GetSize ( ) 는 RectangleShape의 너비와 높이의 값을 돌린다. GetWindow()에 대한 선언은 RectangleShape를 포함하는 SimpleWindow에 대한 참조를 돌린다는것을 의미한다. 다시 말하여 Shape를 포함하는 창문의 복사를 하지 말아야 하기때문에 이러한 방법을 리용한다.

또한 매 검토자가 자기의 선언부분에 예약어 const를 가진다는데 주의하여야 한다. 례를 들면 검토자 GetColor()의 선언은 성원함수가 상수함수라는것을 의미한다.

```
color GetColor() const;
```

즉 이 함수는 객체를 수정하거나 바꿀수 없다. 함수가 객체를 변화시킬수 없다는데로부터 그 코드를 읽을수 있으며 그로부터 콤파일러가 객체의 값을 잘못 바꾸지 않는다는것을 확인할수 있게 한다. 만일 상수함수가 객체의 값을 바꾸려고 하거나 비상수함수에 접근하려고 한다면 콤파일러는 오유를 통보한다.

검토자의 선언뒤에 변이자의 선언이 놓인다. 변이자성원함수는 SetColor(), SetPosition(), SetSize()이다. 변화될수 있는 매개의 속성들은 하나의 변이자를 가진다. 례를 들어 코드

```
R2.SetColor(Blue);
```

는 R2에 인수 Blue를 가진 색설정통보문을 보낸다. R2는 이 통보문을 받으면 자기의 색속성을 Blue로 설정한다. 코드

```
R2.SetPosition(2.5, 6.0);
R2.SetSize(1.0, 4.0);
```

은 R2의 다른 속성들을 설정하기 위하여 통보문을 보낸다. 첫 통보문은 R2의 중심이 창문의 왼쪽변두리로부터는 2.5cm, 창문의 웃쪽변두리에서부터는 6cm 떨어 진 곳에 있다는것을 지적한다. 다음통보문은 R2에 그의 너비가 1.0cm, 높이가 4.0cm라는것을 지적한다. 여러가지 속성들을 설정한후에 그리기 통보문을 보내여 창문에 R2를 현시할수 있다. 호출

```
R2.Draw();
```

는 R2를 그의 창문에서 x, y자리표에 그의 색갈과 크기로 그리라는 통보문을 보낸다. 자료성원의 구체례만들기와 초기화, R2의 그리기처리를 그림 7-4에 보여 주었다.

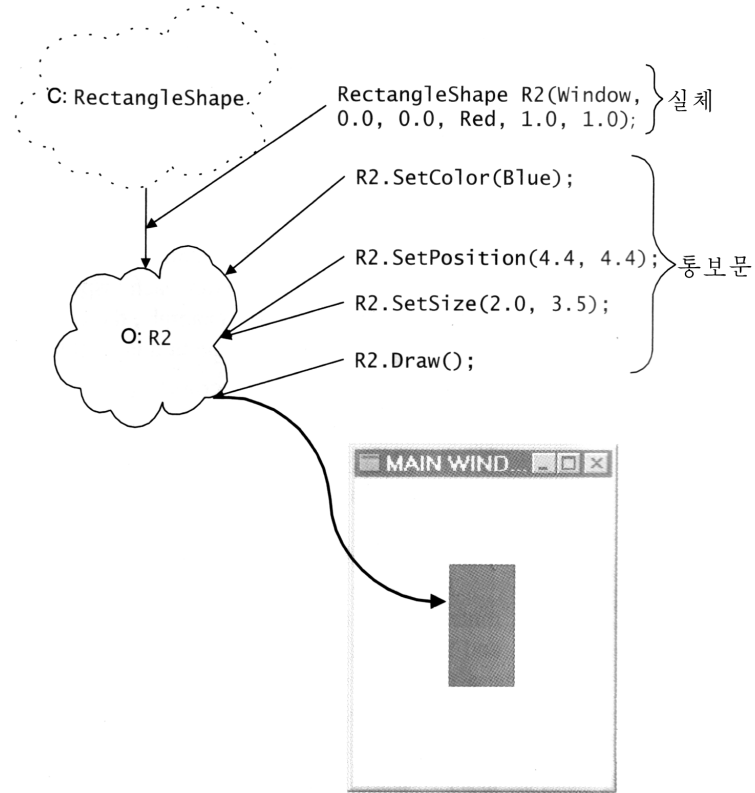



그림 7-4. RectangleShape R2의 구체례만들기, 초기화, 그리기

7.2.1 public 와 private 접근

1 장에서 언급하였던 쏘프트웨어공학의 원리의 하나는 정보은페 즉 교갑화였다. 이러한 처리가 객체를 외부적측면과 내부적측면으로 갈라 놓는다는것을 상기하시오. 객체의 외부적측면은 그 체계의 다른 객체가 볼수 있게(알게)해야 하는것들이다. 객체의 내부적측면은 그 체계의 다른 부분들에 영향을 주지 말아야 하며 따라서 체계의 다른 객체들에 의해 공개되거나 리용가능하게 되지 말아야 하는것들이다. 클라스선언안에 있는 public 와 private 표식은 정보은페 혹은 교갑화를 지원하기 위하여 C++가 제공하는 예약어들이다.

알림

private 보다 앞선 public 선언

C++클라스선언에서 public와 private부분이 임의의 순서로 나타날수 있다. 일부 프로그람작성자들은 처음에 private부분을 놓는데 그것은 자료속성들을 포함하는 부분이기때문이다. 여기서는 public부분이 먼저 놓인다고 보는데 그것은 public성원함수들과 구축자들이 클라스에 대한 사용자대면부이기때문이다. 실지로 private부분에서 자료속성들은 클라스의 리용에서 어느 정도 부차적이며 적당하게 정의된 클라스에서는 자료속성들이 어떻게 선언되였는가를 알 필요가 사실상 전혀 없다.

private표식뒤에 놓이는 자료성원들과 성원함수들은 자기 클라스의 다른 성원함수들에만 접근할수 있다. public표식뒤에 놓이는 클라스성원들은 그 클라스의 성원이 아닌 함수들에도 접근할수 있다. C++클라스의 성원들에 대한 접근조종의 개념을 그림 7-5에서 보여 주었다.

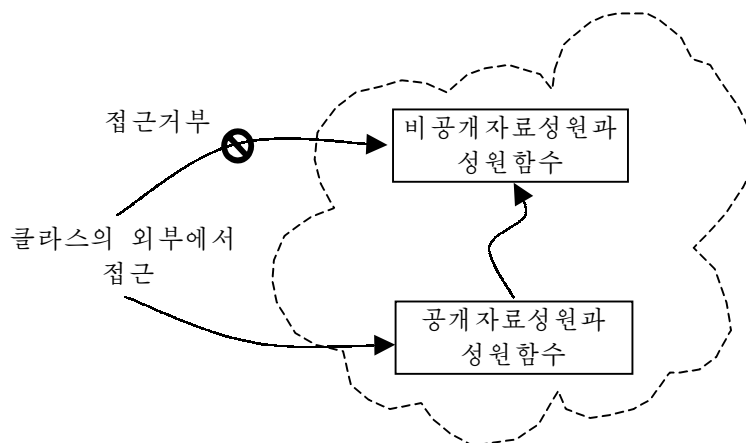



그림 7-5. 공개부분과 비공개부분을 가진 클라스성원들에로의 접근조종

비록 public와 private부분들이 자료성원들과 성원함수들에 대한 접근조종에 리용될수 있지만 접근지정자의 보다 중요한 리용은 정보은페화를 실현한다는데 있다. 정보은페의 원리에 의하면 객체의 모든 호상작용은 기초객체실현세부들을 무시할수 있게 하는 잘 정의된 대면부를 통하여 이루어 진다. 이렇게 되여 public부분에서 성원함수들과 자료성원들은 객체에 대하여 외부적인 형태를 이루며 private부분의 항목들은 객체리용에서 호출할 필요가 없는 객체의 내부부분이다.

RectangleShape클라스의 성원함수들인 Draw(), GetColor(), GetSize(), GetPosition(), setWidth(), GetHeight(), GetWindow(), SetColor(), SetPosition(), SetSize()는 Rectangle Shape에서 public대면부이다. RectangleShape의 public대면부에 의해 R라는 RectangleShape객체를 정의하고 R의 속성들을 설정하기 위하여 적당한 성원함수들을 호출하여 창문안에서 하나의 4각형을 창조하는 코드를을 쓸수 있었다. RectangleShape가 어떻게 표현되고 완성되는가 하는 정확한 서술은 필요하지 않다.

례를 들어 RectangleShape의 public대면부는 류동소수점객체에 보관된 cm값을 리용하여

RectangleShape의 크기와 위치를 정한다는것을 말해 준다. 그러나 기본창문체계는 cm가 아니라 화소에 기초한 체계이다.

RectangleShape의 **public**대면부에 의해 도형들의 위치와 크기를 결정하는데서 보다 편리한 측정체계를 리용할수 있다. 더우기 우리는 창문들이 현시되는 기초자리표계에 대하여 아는것이 없다. 교갑화의 매력은 적합치 않고 복잡한 세부들을 숨김으로써 객체의 리용을 간단하게 한다는데 있다.

C++언어

기정접근

객체가 앞에 **public** 와 **private** 표식이 없이 선언될 때 그 객체들에 대한 기정접근은 **private**이다. 례를 들어 클라스선언

```
class Obj {
        int x;
        int y;
        public:
        Obj();
};
```

에서 옹근수객체 x와 y는 그 선언부앞에 접근표식이 없으므로 **private**접근으로 된다.

문 제

1. 속성값을 돌려 주는 성원함수를 무엇이라고 부르는가?
2. 속성값을 변경시키는 성원함수를 무엇이라고 부르는가?
3. 일부 기능이나 봉사를 수행하는 성원함수를 무엇이라고 부르는가?
4. 객체를 창조하는 성원함수를 무엇이라고 부르는가?
5. 객체를 창조하고 그 자료성원들을 초기화하는 처리를 무엇이라고 하는가?
6. RectangleShapeBigBox를 록색으로 설정하는 명령을 쓰시오.
7. 4cm의 너비와 3cm의 높이로 RectangleShapeButton의 크기를 설정하는 명령을 쓰시오.
8. 창문의 꼭대기로부터 2cm, 왼쪽으로부터 4cm되는 곳에 RectangleShapeDoor의 위치를 정하는 명령을 쓰시오.
9. 다음과 같은 클라스선언을 고찰하시오.

```
class Person {
    public:
        person();
        string GetName();
    private:
        bool Citizen();
        string LastName;
        string FirstName;
        string PhoneNumber;
};
```

다음의 코드토막에서 무엇이 틀리는가를 설명하시오.

```
int main() {
```

```
    Person P;
    ...
    string Pnumber = P.PhoneNumber;
    return 0;
}
```

10. 아래와 같은 클라스선언이 있다.

```
class Person {
        Person();
        string GetName();
    private:
        bool Citizen();
        string LastName;
        string FirstName;
        string PhoneNumber;
};
```

검토자 GetName()에 대한 접근은 public인가, private인가?

7.3 RectangleShape 클라스의 리용

지금까지 클라스의 객체들을 구체레화하며 그 성원함수들을 통하여 객체와 호상작용하여 클라스선언의 기초지식을 리해하였으므로 더 자세하게 클라스 RectangleShape의 리용을 서술할수 있다. 상기할것은 창문에서 4각형들의 위치를 결정하는데 리용되는 자리표계는 그 원점이 창문의 왼쪽웃모서리에 있다는것이다. 그림 7-6에 SimpleWindow의 자리표계를 보여 주었다.

자리표축들의 단위는 cm이다. 정의

```
RectangleShape T(ExampleWindow,2,3, Red,2,2);
```

로부터 그 중심이 창문의 왼쪽으로부터 2cm, 꼭대기로부터는 3cm 되는 곳에 있는 T라는 Rectangle Shape를 창조한다.

위치 혹은 크기를 가지게 되는 RectangleShape의 성원함수에 대하여 모든 인수들의 척도는 cm이다. 하나의 객체의 클라스에 대하여 시종일관 일정한 대면부를 가지는것은 객체지향설계의 중요한 원리이다. 례를 들어 다음의 프로그람은 1cm의 폭과 너비, 그리고 창문 w의 왼쪽과 꼭대기로부터 각각 3cm 되는 곳에 위치한 푸른색4각형 B를 창조하고 그린다. 그다음에 프로그람은 4각형 M을 창조한다. M의 너비와 높이는 2cm이며 그의 색갈은 단홍색으로 설정되고 창문 W에 현시되도록 되여 있다.

M의 위치는 그의 오른쪽아래구석이 B의 왼쪽웃구석에 놓이도록 자리표 (2.5, 2.5)로 설정된다.

```
#include "rect.h"
SimpleWindow W("Double Square", 8, 8);
int ApiMain() {
    W.Open();
```

```
    RectangleShape B(W, 4, 4, Blue, 1, 1);
    B.Draw();
    RectangleShape M(W, 2.5, 2.5, Magenta, 2, 2);
    M.Draw();
    return 0;
}
```

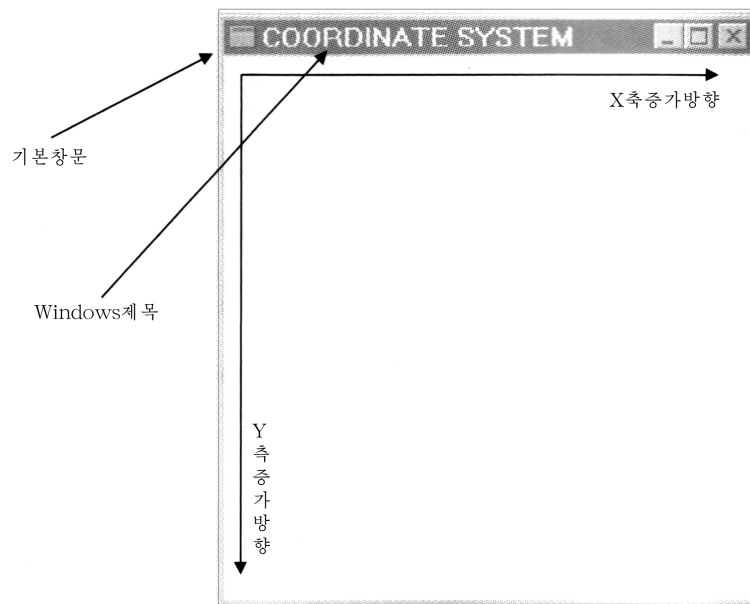



그림 7-6. SimpleWindow자리표계

그림 7-7에 결과창문을 보여 주었다.

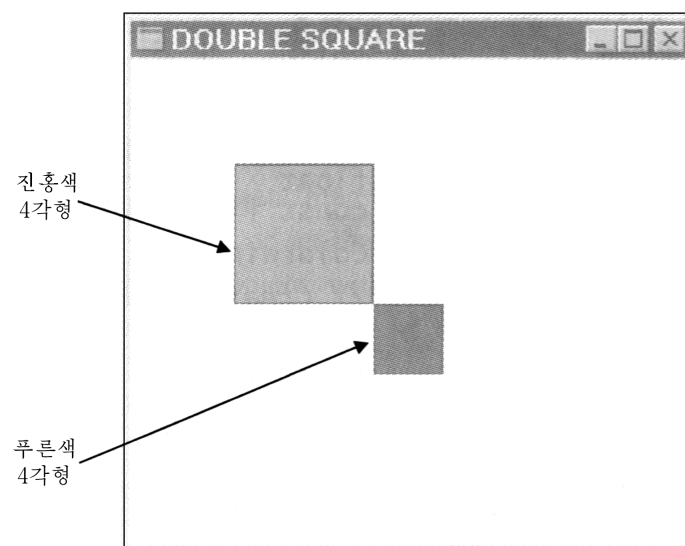



그림 7-7. 2개의 바른4각형이 현시된 창문

창문체계의 다른 중요한 특징은 색체계를 리용할수 있다는것이다. 그 체계를 때로는 조색판이라고 한다. RectangleShape객체가 가질수 있는 모든 색갈을 현시하는 Colors라는 프로그람을 작성해 보자.

렬거형으로 RectangleShape객체의 색갈을 지정할수 있다.

```
enum color{Black, White, Red, Green, Blue, Yellow, Cyan, Magenta};
```

이제 6개의 바른4각형 즉 Black와 White를 제외한 색갈에 대한 4각형들을 현시하려고 한다. White는 배경색과 같으므로 그릴 필요가 없다. 화면이 색스펙트르와 류사하도록 매변이 1cm로 된 바른4각형을 6개의 색갈로 현시하여 변끼리 붙여 배렬한다. 물론 창문안에서 색스펙트르가 중심에 놓이도록 하여 그것이 잘 보이게 하여야 한다.

프로그람 7-1에 이 프로그람에 대한 코드화가 들어 있다. 처음으로 몇개의 상수들이 4각형의 크기와 첫번째 색갈의 위치를 지정한다. 그 정의에 따라 RectangleShape객체 ColorPatch가 구체레화된다. for명령은 색갈들이 순환하게 한다. **for**순환의 증가표현식은

```
c= (color) (c +1 )
```

이다. 이 명령은 c를 다음색갈로 설정한다. 왜 다음색갈로 바꾸는데 ++C를 쓰지 않는가고 물을수도 있다. C++는 렬거형객체들에 대해서는 증가연산자를 쓰지 않도록 한다. 그 리유는 렬거형의 한 성원이 주어 졌을 때 다음성원이 다음옹근수에 의하여 표현될수 없기때문이다. 그러나 렬거형의 기초실현이 옹근수이므로 객체들에 대하여 산수적인 렬거를 수행할수 있다. 그 결과는 옹근수이다. 색성원들이 련속적인 옹근수들이라는것을 알기 때문에 다음색갈을 얻기 위하여 표현식 C+1의 결과를 color에로 넘겨도 된다. 이 방법은 자연스럽게 **for**순환을 작성할수 있게 한다.

for명령문의 본체는 객체 ColorPatch를 적당한 색갈로 설정하여 칸을 그리고 그 위치를 다시 설정하여 다음칸이 린접하여 나타나게 한다. 그림 7-8은 프로그람에 의해 창조된 결과창문을 보여 준다.

```
#include " rect.h"
SimpleWindow ColorWindow(" color Palette", 8.0, 8.0);
int ApiMain() {
    const float SideSize = 1.0;
    float Xposition = 1.5;
    const float Yposition = 4;
    ColorWindow.Open();
    RectangleShape ColorPatch(ColorWindow,
      Xposition, Yposition, White, SideSize, SideSize);
    for (color c = Red; c <= Magenta; c = (color) (C +1)) {
        ColorPatch.SetColor(c);
        ColorPatch.SetPosition(Xposition, Yposition);
        ColorPatch.Draw();
        Xposition += SideSize;
    }
    return 0;
}
```

프로그람 7-1. 색문양을 창조하고 현시하는 프로그람

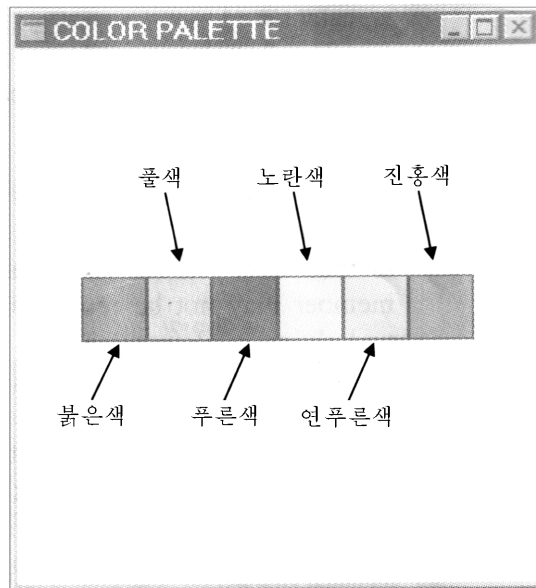



그림 7-8. 조색판프로그람에 의해 현시된 창문

7.4 구축자

지금까지의 실례들에서 매번 RectangleShape를 정의할 때마다 모든 자료성원들에 초기값을 준것은 RectangleShape구축자가 그것들을 요구하였기때문이다. 특히 조색판프로그람에서 한것처럼 직접 자료성원값들을 바꾸거나 설정하려고 할 때에는 그 모든 자료성원값을 지적하지 않고 클라스형객체를 창조해야 한다. 그러나 속성들에 대한 값을 지적하지 않고 객체를 창조하면 위험하다. 례를 들어 RectangleShape를 창조하고 그에 위치를 주지 않은 상태에서 그자체를 그리라는 통보문을 보내면 프로그람에 의해 체계가 파괴되는것을 비롯하여 예측할수 없는 이상한 현상들이 발생할수 있다. C++는 요구하는것을 정확히 수행할수 있는 방법을 제공하여 준다.

6장에서 언급하였던 기정파라메터들을 상기해 보자. 구축자에서도 이러한 기구구조를 리용할수 있다. 다음과 같이 RectangleShape구축자의 원형을 바꿀수 있다.

```
RectangleShape(SimpleWindow &Window, float Xcoord = 0,
  float Ycoord = 0,  const color &c = Red,
  float Width  = 1.  float Height = 2);
```

구축자에서 기정파라메터값들의 리용은 함수원형에서 기정파라메터값을 리용할 때와 꼭같이 작용한다. 례를 들어 정의

```
RectangleShape C(ControlWindow);
```

은 C라는 RectangleShape를 창조하며 그의 SimpleWindow속성을 ControlWindow로 설정한다. 다른 속성들은 구축자원형에서 지적된 기정값들로 설정된다. 이리하여 위치속성들 XCenter, YCenter는 값 0.0 으로 주어 지며 자료성원 Color는 Red로, 그리고 자료성원들인 Width와 Height는 각각 값 1.0과 2.0으로 설정된다.

마찬가지로 정의

```
RectangleShape D (ControlWindow, 3, 5) ;
```

은 창문 ControlWindow의 왼쪽변두리로부터 3cm, 웃쪽꽉대기로부터는 5cm 되는 위치에 중심을 가진 D라는 RectangleShape를 구체례화한다. 정의에서 색과 크기, 속성값들을 지적하지 않았으므로 Red, 1.0과 2.0이라는 기정값이 설정된다.

인수 Window는 기정값을 가지지 않는다. 따라서 정의

```
RectangleShape E;
```

은 틀리는 코드화이며 C++번역프로그람은 참조자료성원들이 객체가 구축될 때 초기화되여야 하므로 그것이 오유라고 통보한다. 일단 참조자료성원들이 초기화되면 그것은 변경할수 없다.

주의

EzWindows 원천코드

만일 RectangleShape클라스에 대한 대면부와 그의 실현에 대해 조사하고 싶다면 그 대면부(rect.h)와 실현부(rect.cpp)를 이 책에 따라 나오는 CD상에서 찾아 볼수 있다. rect.h를 비롯한 모든 EzWindow형태에 대한 대면부파일은 등록부

{BaseDirectory}\ezwin\include

에서 찾아 볼수 있다. 마찬가지로 대응한 도형들에 대한 실현부파일은 등록부

{BaseDirectory}\ezwin\src

에서 찾아 볼수 있다. 두 경우에 {BaseDirectory}는 EzWindow가 설치된 하드구동기상의 위치이다.

EzWindow software의 제일 마지막판본은 다음의 Web싸이트에서 내리적재된다.

http: //www.cs.virginia.edu/C++programdesign

문 제

11. 날자객체에 대한 적당한 C++클라스를 작성하시오. 클라스 Date는 다음의 정보를 포함하게 된다.

- 년
- 월
- 일

클라스선언은 정보은페화원리를 리용하여 작성하시오. 즉 검토자와 변이자를 가져야 한다. Date에 대한 단일구축자는 다음과 같이 Date객체구조를 지원한다.

- 특수한 날자는 주어 지지 않는다. 이 경우에 다른 자료마당은 모두 1로 초기화된다.
- 년만이 주어 진다. 이 경우에 다른 자료마당들은 1로 초기화된다.
- 년과 월만이 주어 진다. 이 경우에 날자는 기정값 1로 된다.

12. 실례로 다음의것들은 Date객체들의 합법적인 선언부들이다.

```
Date Date1;
Date Date2(1999);      //년을 1999로 설정
                       //월을 1월로 설정
                       //날자를 1일로 설정
DateDate3(1999,1);     //년을 1999로 설정하고
                       //월을 1월로 설정하고
```

```
                         //일을 1로 설정한다.
DateDate4(2000,2,13);//년을 2000으로 설정
                         //월을 2월로 설정
                         //일을 13일로 설정
```

클라스선언에는 지정된 구축자(구축자는 오직 하나이다.), 검토자, 변이자외에 그 어떤 성원함수도 포함할 필요가 없다.

13. 책객체에 대한 적당한 C++클라스선언을 하시오. 클라스 Book는 다음과 같은 자료들을 포함한다.

- 책제목
- 저자들(MaxAuthors가 상수일 때 최대라고 가정한다.)
- 유일식별번호(옹근수로 가정한다.)

클라스선언에 정보은폐화원리를 리용한다. 즉 적합한 검토자와 변이자가 있다. 풀이방향: 저자자료에 필요한 검토자와 변이자에 주의를 돌리시오. 이 자료성원에 대하여서는 오직 하나의 검토자와 변이자만이 있다. Book에 대한 하나의 구축자는 다음과 같은 경우 Book객체구조를 지원한다.

- 제목이 지적되고 저자, 식별번호가 주어 진다.
- 제목과 저자가 주어 지지만 식별번호가 주어 지지 않는다. 이 경우에 식별번호는 기정값 0으로 된다.
- 제목이 주어 지지만 저자와 식별번호가 주어 지지 않는다. 이 경우에 저자는 문자렬 " NO Author"이고 식별번호는 0으로 된다.
- 제목, 저자 혹은 식별번호가 주어 지지 않는다. 이 경우에 제목은 기정값 " No title"로 되며 다른 항목들은 우와 같이 설정된다.

실례로 다음의 선언은 옳다.

```
Book Book1(" My Friend Linda", :Monica Lewinsky", 2);
Book Book2(" Eat a Peach", " Greg Allman");
Book Book3(" My Life As a Dog");
Book Book4;
```

클라스선언은 필수적인 구축자와 검토자, 변이자를 제외한 임의의 다른 성원함수들을 포함할 필요가 없다.

14. 위도를 표현하는데 리용되는 Latitude라는 객체의 구축자실현부와 완전머리부 c.h파일을 정의하시오. 클라스는 객체지향설계기술에 의하여 설계되고 완성된다. 하나의 속성은 클라스가 정보은폐화를 지원한다는것이다. 위도는 4개 값 즉 도, 분, 초 그리고 그 지역이 북쪽인가, 남쪽인가로 지정한다.이렇게 클라스 Latitude는 정확한 3개의 옹근수자료성원들 즉 도, 분,초에 대한 옹근수를 가지며 북쪽인가, 남쪽인가 하는 하나의 자료성원을 가진다. 북쪽/남쪽지시자는 머리부파일에서 선언하여야 할 렬거형을 리용하여 보관한다. 표준검토자와 변이자외에 클라스는 구축자를 가진다. 그 구축자는 4개의 자료성원에 대한 기정값들을 제공한다. 례를 들어 선언

```
Latitude Equator;
```

은 매개의 옹근수자료성원들을 0으로 설정하고 북쪽/남쪽지시자를 North로 설정하여 Latitude객체

를 창조한다. 코드화를 될수록 효과적으로 하시오.

15. EzWindows에 의해 지원되는 색갈들을 현시하는 프로그람을 설계하고 완성하시오. 프로그람은 8cm의 높이와 5cm의 너비로 SimpleWindow를 구축한다. 창문에서 그 아래부분에 매 색갈의 높이가 1cm이고 너비가 5cm인 줄무늬를 그리시오.

7.5 만화경프로그람의 설계

클라스 RectangleShape의 마지막실례로서 창문에 만화경현시물을 나타내는 프로그람을 설계하고 완성해 본다. 만화경을 모의하기 위한 프로그람의 첫 실현부에서 객체들을 리용하여야 하지만 프로그람설계가 객체지향적이지 못하다. 13장에서는 만화경을 모의하는 개선된 프로그람을 주었다. 이 프로그람은 객체지향적인 방법을 리용하여 설계되였다.

1816년에 다비드 브레비스터에 의해 발견된 만화경은 그 크기와 도형이 망원경과 어딘가 비슷하며 그의 대안경을 통하여 여러개의 색갈로 된 대칭문양을 창조한다. 아이들이 가지고 장난하는 만화경은 마분지관으로 만들어 져 있다. 관은 색을 입힌 유리와 거울쪼각들을 포함하고 있으며 다른 관은 화상을 보이게 하는 대안렌즈로 되여 있다. 화상은 색을 입힌 유리쪼각들이 반사조합의 결과가 얻어 진다. 그 문양은 색유리와 거울을 가진 관을 회전시켜서 바꿀수 있다. 색유리의 쪼각들이 움직일 때 새로운 문양을 만들어 낸다.

만화경화상은 창문에 서로 대칭적으로 위치하는 화상들로서 모의할수 있다. 만화경프로그람의 첫 판본에서 현시되는 화상들은 여러가지 크기와 색갈을 가지는 바른4각형들이다.

각이한 크기와 색갈을 가진 장식품들을 대칭적으로 현시하여 만화경효과를 창조하는 프로그람 Kaleidoscope를 작성해야 한다. Kaleidoscope()은 현시창문에 새로운 장식품을 추가하기 위하여 반복호출된다. 대칭적인 문양을 만드는 공정은 4개의 구역으로 창문을 가르는것이다(그림 7-9를 보시오). 거울형반사효과를 내기 위하여 같은 장식품들을 반대켠 분구들에 배치하였다. 4개의 장식품들이 창문중심에 대하여 서로 대칭되게 배치되였다. 그물효과는 반대켠 분구에 있는 장식품들을 반사되는것 같이 보이게 하는것이다. 4개의 첨부된 장식품들은 혼합한 색방식과 비슷하게 놓이게 된다. 이렇게 총 8개의 장식품들은 매 분구에 2개씩 그려 진다.

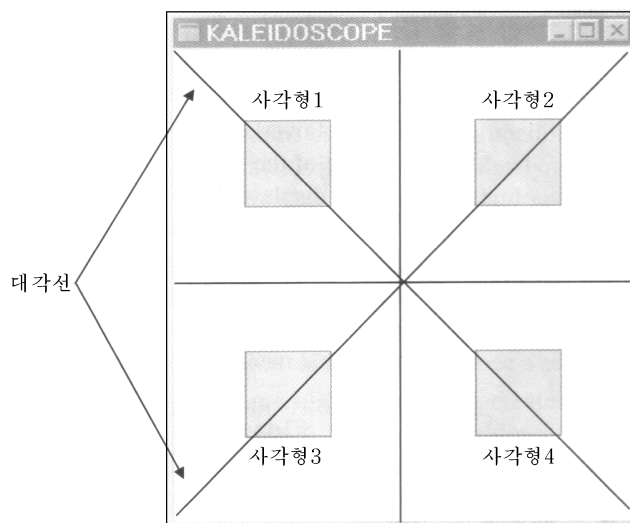



그림 7-9. 만화경을 모의한 창문

진행해야 할 3개의 보조과제는 색갈, 장식품의 크기, 그리고 창문중심으로부터 장식품까지의 편위를 얻는것이다. 이 모든 공정은 5.7에서 제기한 함수 Uniform()을 리용하여 설계할수 있다. Uniform()과 형변화연산자를 리용하여 우연색을 쉽게 얻을수 있다. 다음의 함수는 하나의 우연색갈을 생성한다.

```
color RandomColor() {
    return ( color ) Uniform(1, MaxColors -1);
}
```

귀환명령문에서 Uniform()에 의하여 돌려 지는 우연번호가 color형으로 형변환을 시켰다는데 주의하시오. 이 방법은 color가 렬거형이며 그의 기초실현부가 0으로부터 시작되는 옹근수들의 렬일 때에만 적용된다. 이렇게 1을 붉은색으로, 2는 록색으로, 3은 푸른색 등으로 배치한다. 형변환은 Uniform()에 의하여 돌려 진 옹근수를 색갈로 변환시킨다는것을 말해 준다.

장식품의 우연적인 크기를 생성하기 위하여 우와 같은 방법으로 할수 있다. 우리가 취한 어떤 최대값과 1사이의 어떤 크기를 생성한다고 하자. 장식품의 크기가 창문의 크기보다 더 커서는 안되기때문에 최대크기를 조종해야 한다.

다양한 현시물을 얻기 위하여 장식품크기의 범위를 제한하고 아주 작은 크기를 가지도록 해야 한다. 시작위치로부터 0.1cm의 증분으로 1로부터 최대크기까지의 장식품을 생성한다. 함수

```
float RandomTrinketSize(int MaxSize) {
    return Uniform (10, MaxSize = 10) /10.0;
}
```

는 1에서부터 MaxSize까지의 범위에서 0.1cm의 증가량으로 MaxSize까지의 수들을 생성한다. 함수는 1, 1.1, 1.2, …, MaxSize까지의 수들중에서 1개를 돌린다. 또한 대각선을 따라 장식품을 배치하기 위한 우연적인 편위위치를 생성하여야 한다. 편위를 생성하는것은 장식품이 그려 질 때 그것이 창문중심에 덧놓이지 않게 편위를 충분히 크게 할 필요가 있으므로 좀 어렵다. 편위는 장식품의 크기에 의존한다. 편위가 너무 크면 바른4각형부분이 창문밖에 현시된다. 장식품의 크기를 고려하기 위하여 편위가 최소한 장식품의 한변의 절반만해야 한다.

이렇게 편위를 최소한도의 길이로 하면 장식품이 그려 질 때 중심에 놓이지 않게 된다고 확신할수 있다. 다음과 같은 함수는 요구에 맞는 임의의 편위위치를 생성한다.

```
float RandomOffset(int Range, float TrinketSize ) {
    float offset = (Uniform(0, Range * 10)) / 10.0;
    if (offset < (TrinketSize)/2))
        offset = TrinketSize / 2;
    return offset;
}
```

이제는 Kaleidoscope()를 작성할수 있는데 그 기능에 의해 요구되는 8개의 장식품들을 그릴수 있다. 다음의 코드화는 첫 4개의 장식품들을 창조한다.

```
const float Center = WindowSize /2.0;
float TrinketSize = RandomTrinketSize(Maxsize);
float Offset = RandomOffset(TrinketSize, MaxSize);
```

```
color FirstColor = RandomColor();
color SecondColor = Randomcolor();
//4개의 장식품을 창조한다.
RectangleShape TrinketQuad1(KaleidoWindow,
  Center + Offset, Center + Offset, FirstColor,
  TrinketSize, TrinketSize);
RectangleShape TrinketQuad2(KaleidoWindow,
  Center - Offset, Center + Offset, SecondColor,
  TrinketSize, TrinketSize);
RectangleShape TrinketQuad3(KaleidoWindow,
  Center - Offset, Center - Offset, FirstColor,
  TrinketSize, TrinketSize);
RectangleShape TrinketQuad4(KaleidoWindow,
  Center + Offset, Center - Offset, SecondColor,
  TrinketSize, TrinketSize);
```

코드화의 첫 부분은 장식품을 창조하는데 필요한 여러가지 값들을 계산한다. 첫행은 바른4각형창문의 중심자리표를 계산한다. 다음행은 장식품의 크기를 계산하고 적절한 함수에 접근하여 창문의 중심으로부터의 편위를 계산한다. 설치코드의 마지막비트는 장식품들에 대한 2개의 색갈을 준다. 이 정보를 리용하여 매 분구에서의 장식품들이 구체레화된다. 일단 장식품들이 구체레화되면 다음의 코드화는 장식품들을 그린다.

```
TrinketQuad1.Draw();
TrinketQuad2.Draw();
TrinketQuad3.Draw();
TrinketQuad4.Draw();
```

다음과제는 장식품들을 보관하였다가 새로운 위치에 그것들을 그리는것이다. 새로운 편위를 얻기 위하여 RandomOffset()함수를 접근한다. 이 편위를 리용하여 장식품들이 RectangleShape의 SetPosition()변이자에 의해 보관된다. 다른 4개의 바른4각형들을 그리기 위한 코드도 우의 코드와 같다. 프로그람 7-2는 완성된 Kaleidoscope프로그람을 포함한다.

```
//프로그람 7-2: 간단한 만화경을 모의한다.
#include " uniform.h"
#include " rect.h"
//창문의 크기
const float WindowSize = 10.0'
//장식품의 최대크기
const float MaxSize = 4.0;
//4각형의 창문을 창조한다.
SimpleWindow KaleidoWindow(" Kaleidoscope",
```

```
    WindowSize, WindowSize);
    color RandomColor() {
      return (color) Uniform(0, MaxColors - 1);
  }
  //RandomOffset-창문의 중심에서 장식품까지의 란수적인 편위주소를 발생한다.
  float RandomOffset(int Range, float TrinketSize ) {
      float Offset = (Uniform(0, Range * 10)) / 10.0;
      if (Offset < (TrinketSize)/2))
          Offset = TrinketSize / 2;
      return Offset;
  }
  float RandomTrinketSize(int MaxSize) {
      return Uniform(10, MaxSize *10)/10.0;
  }
  int Kaleidoscope() {
      const float Center = WindowSize /2.0;
      float TrinketSize = RandomTrinketSize(Maxsize);
      float Offset = RandomOffset(TrinketSize, MaxSize);
      color FirstColor = RandomColor();
      color SecondColor = Randomcolor();
      RectangleShape TrinketQuad1(KaleidoWindow,
        Center + Offset, Center + Offset, FirstColor,
        TrinketSize, TrinketSize);
      RectangleShape TrinketQuad2(KaleidoWindow,
        Center - Offset, Center + Offset, SecondColor,
        TrinketSize, TrinketSize);
      RectangleShape TrinketQuad3(KaleidoWindow,
        Center - Offset, Center - Offset, FirstColor,
        TrinketSize, TrinketSize);
      RectangleShape TrinketQuad4(KaleidoWindow,
        Center + Offset, Center - Offset, SecondColor,
        TrinketSize, TrinketSize);
      TrinketQuad1.Draw();
      TrinketQuad2.Draw();
      TrinketQuad3.Draw();
      TrinketQuad4.Draw();
      TrinketQuad1.SetColor(SecondColor);
      TrinketQuad2.SetColor(firstColor);
      TrinketQuad1.SetColor(SecondColor);
```

```
        TrinketQuad1.SetColor(firstColor);
        Offset = RandomOffset(MaxSize, TrinketSize);
        TrinketQuad1.SetPosition(Center, +Offset,
          Center + Offset);
        TrinketQuad2.SetPosition(Center, -Offset,
          Center + Offset);
        TrinketQuad3.SetPosition(Center, -Offset,
          Center - Offset);
        TrinketQuad4.SetPosition(Center, +Offset,
          Center - Offset);
        TrinketQuad1.Draw();
        TrinketQuad2.Draw();
        TrinketQuad3.Draw();
        TrinketQuad4.Draw();
        return 0;
    }
    int ApiMain() {
        InitializeSeed();
        KaleidoWindow.Open();
        KaleidoWindow.SetTimerCallback(Kaleidoscope);
        KaleidoWindow.StartTimer(1000);
        return 0;
    }
```

프로그람 7-2. 간단한 만화경

함수 ApiMain()은 새로운 서고함수들에 대한 호출을 포함하고 있다. Kaleidoscope가 천천히 이동하는것처럼 보이게 하기 위하여 함수 Kaleidoscope()을 주기적으로 호출해야 한다. SimpleWindow클라스는 이러한 목적을 위하여 특수하게 설계된 시간계수기객체를 가지고 있다. SimpleWindowSetTimerCallBack()성원함수는 시간계수기를 설정하며 시간계수기가 동작을 끝마치면 Kaleidoscope함수를 호출하기 위한 EzWindows체계를 호출한다. SimpleWindow성원함수 StartTimer()는 시간계수기를 기동하며 1000ms 혹은 1s마다 동작하도록 한다.

이렇게 함수 Kaleidoscope()는 실제적인 Kaleidoscope에 의해 창조된 화상이 아주 많은것처럼 보이도록 한다. 그림 7-10은 프로그람을 30초동안 실행한후 창조된 화상을 보여 준다.

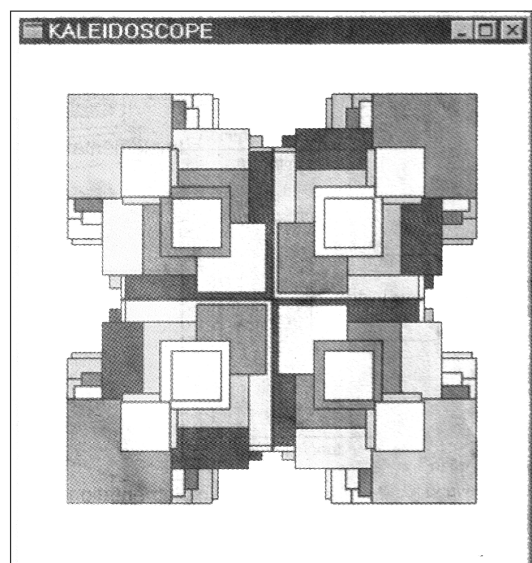



그림 7-10. Kaleidoscope 함수를 여러번 호출하여 창조한 만화경창문

7.6 객체지향분석과 설계

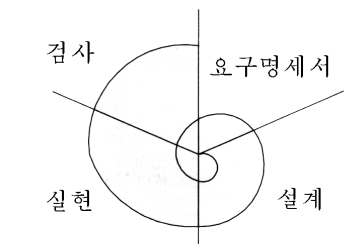
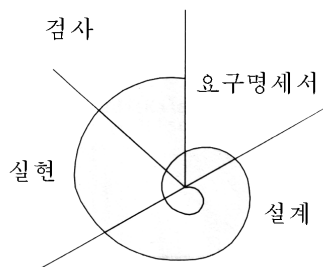
클라스구조는 객체지향설계를 실현하기 위한 중요한 단계이다. 객체지향설계는 대부분의 중요한 학문들과 마찬가지로 정통하기가 힘들다. 그러나 이 설계는 쏘프트웨어의 갱신원가를 줄이는데 도움을 준다. 그림 7-11은 객체지향적인 방법과 전통적인 쏘프트웨어개발과의 차이를 설명한다. 전통적인 쏘프트웨어개발방법에서는 완성과 시험이 개발시간의 가장 큰 부분을 차지한다. 객체지향수법에서는 설계가 총 개발시간의 가장 큰 부분을 차지한다. 객체지향설계에서는 잘된 설계가 실현하기도 쉽고 오유도 더 적기때문에 실현과 유지에 소비되는 시간이 작다. 그러나 시간의 더 큰 비중은 체계를 재리용하고 유지와 확장성에 맞게 설계하기 어려우므로 체계를 설계하는데 돌려 지게 된다. 비록 두가지 방법들이 총체적으로는 같은 개발시간을 요구한다고 해도 유지하는데 원가가 적게 드는 객체지향방법을 리용하여 설계된 쏘프트웨어는 확장하기가 쉽고 그의 일부분은 다른 응용프로그람들에서 재리용될수 있다. 따라서 이러한 품에 드는 시간은 대단히 줄어 든다.

전통적인 쏘프트웨어생명주기

객체기지향쏘프트웨어생명주기

그림 7-11. 객체지향쏘프트웨어계획과 그 생명주기의 비교

객체지향분석(OOA)과 객체지향설계(OOD) 그리고 클라스들을 리용한 객체지향설계의 실현에 대한 기초를 설명하기 위하여 실제적인 응용프로그람들에 대한 OOA와 OOD로 들어 가 보자.

대부분의 현대적인 공장들은 어디가나 원가를 줄이고 생산공정을 단축하기 위하여 자동화를 실현하였다. 공장자동화로 개선이 있다고 하지만 많은 일감들이 자동화하기가 힘들고 사람들의 손을 요구한다. 례를 들면 사람들이 아직까지도 색갈에서 미묘한 차이를 알아 내고 혼합된 문양을 알아 보는데서 기계보다 훨씬 더 낫다. 과업은 공장에서 특수한 일을 수행하는 사람들을 숙련시킬수 있는 프로그람을 설계하고 완성하는것이다.

이 공장에서 세 종류의 부분품들이 벨트콘베아로 검사위치까지 운반된다. 검사위치에서의 부분품들의 비데오화상은 검열원이 보게 되는 현시장치에로 보내여 진다. 그림 7-12에 검사공정의 조작을 보여주었다.

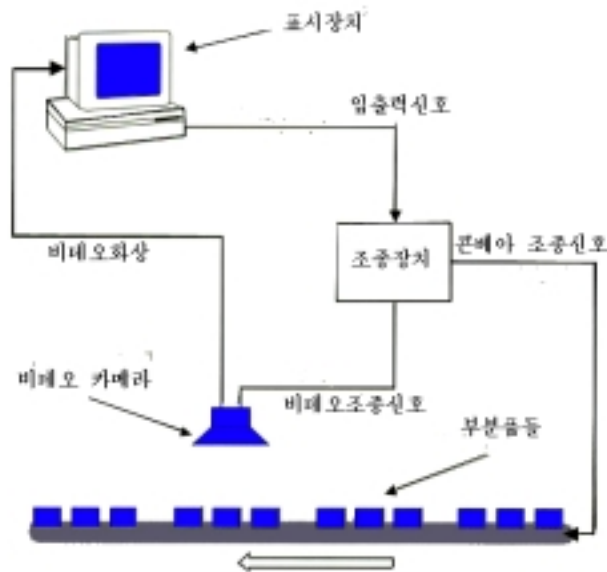



그림 7-12. 공장자동화체계

검열원은 세 부분품의 비데오화상을 검사하여 보고 그것들을 접수할수 있는가, 없는가를 설정한다. 접수할수 있으면 두 부분품은 같은 색갈로 되여야 한다. 만일 두 부분품이 같은 색갈이라면 검열원은 접수단추를 누르고 그 부분품들은 다음단계로 넘어 가 다른 부분품들과 조립된다. 만일 두 부분품이 같은 색갈이 아니라면 검열원은 거절단추를 누르며 세 부분품들은 오물처리장소로 운반된다.

과제는 가능한껏 부분품검열원의 동작을 숙련하고 그것을 평가하는데 쓸수 있는 프로그람을 작성하는것이다. 첨부하면 그 프로그람은 검열원들이 어느 정도 정확하게 부분품들을 결정하는데 도움을 줄수 있다. 이 정보는 부분품이 체계를 통하여 움직이는 속도를 설정하는데 리용되게 된다. 상세하게 서술된 모의프로그람의 설계와 완성을 시작하기에 앞서 이 프로그람이 어떻게 조작을 하여야 하는가에 대한 정확한 설명이 요구된다.

부분품검사모의프로그람은 조립흐름선에서의 부분품의 검사조작을 모의하여야 한다. 부분품들이 벨트콘베아로 움직이는것을 모의하기 위하여 임의로 선택한 색갈로 3개의 4각형들을 창조한다. 현시체계는 콘베아를 타고 나타나는 3개의 4각형들이 현시되는 창문으로 구성된다. 검사원에 의한 입력은 조종창문을 통하여 들어 온다. 검열원은 문자 a을 타자하여(접수하기 위하여) 부분품들을 접수하거나 문자 r를 타자하여(거절하기 위하여) 부분품들을 거절할수 있다. 하나의 부분품모임은 두 부분품의 색갈이 갈다면 접수될수 있어야 한다. 또한 모의프로그람은 검열원의 동작에 대한 통계자료를 가지고 있어야 한다. 그리고 검열원이 조종하는 부분품모임의 수와 검열원이 하는 정확한 결과와 부정확한 결과의 수를 기록하려고 한다. 모의대화가 끌난후에 모의프로그람은 다음과 같은 통계표를 인쇄하려고 한다. 즉 모의조종시간, 그 시간에 검사된 부분품모임의 수, 정확한 결정과 정확치 않은 결정의 수, 그리고 정확한 결정들의 비률 등을 보여 주는 통계표이다.

객체지향설계의 첫 단계는 체계를 구성하는 객체들을 찾고 결정하는것이다. 일반적으로 이 과업은 힘들지 않으며 특히 카메라, 자동현금출납기, 수감장치 혹은 전화와 같은 구조물객체들에 대해서는 더욱 그러하다. 체계에서 리해할수 없는 일부 구체례들을 표현하는 객체들은 결정하기가 더욱 어렵다. 객체들의 구체례들은 비데오화상, 은행거래, 수감장치신호 혹은 전화호출이다.

그림 7-12로부터 객체지향설계에서 실현할 필요가 있는 일부 클라스들을 지적할수 있다. 전반적인

공장자동화체계의 과정이 아니라 모의체계를 작성하고 있기때문에 그림 7-12에 있는 모든 객체들을 반드시 포함할 필요는 없다. 명백하게 비데오카메라, 현시장치, 입력장치와 부분품들이 필요하다. 모의체계를 위해 벨트콘베아를 가정할 필요는 없다. 부가적으로 그림 7-12는 비데오화상, 접수/거절신호, 비데오조종신호와 같은 항목들에 대한 참고를 포함하고 있다. 이 구체례들을 표현하기 위하여 어떠한 객체들을 요구하여야 하는지가 명백하지 않다. 이러한 객체들을 머리속에 구상하여야 하지만 그것들을 아직 클라스 창조에 넘겨서는 안된다.

일단 필요한 객체들을 정하였다면 그것들이 어떻게 호상작용하는가를 보여 주는 도표를 그려 보는것이 좋다. 이 단계는 객체들의 호상작용이 그것들의 동작과 응답에 일정한 영향을 미치기때문에 중요하다. 그림 7-13은 객체들의 호상작용을 보여 주고 있다. 그것은 그림 7-12와 완전히 일치한다. 객체지향분석과 설계의 위력의 하나는 설계를 엄격하고 실제적인 체계로 만드는것이다. 그림 7-13으로부터 비데오카메라가 조종장치에 비데오화상을 보내며 조종장치는 비데오현시장치에 보낸다는것을 알수 있다. 공장자동화체계에 대한 모의체계를 설계하고 있으며 그래서 그 조종장치는 양성생이 어떤 정확한 응답을 하는가를 판단하기 위한 비데오화상을 요구하게 된다. 조종장치는 비데오카메라, 콘베아, 조종탁과 호상작용한다.

객체들이 어떻게 호상작용하는가를 결정한후에 객체들의 동작과 임무에 대하여 생각해 볼수 있다. 비데오카메라의 기능과 속성들을 결정하여 보자. 카메라는 부분품들이 카메라밑을 통과하면 그 부분품의 화상을 얻으므로 그의 임무의 하나는 조종장치의 신호를 받을 때 그 부분품들의 화상을 얻는것이다.

물리적인 장치의 공통적인 속성은 그의 방식 혹은 상태이다. 례하면 사진복사기는 여러가지 상태를 가지는데 LCD판에 차단, 준비상태, 투입, 종이두절상태 그리고 비정상동작 등을 현시하여 준다. 하나의 비데오카메라도 역시 카메라형에 따르는 여러가지 상태를 가진다. 표준적인 상태는 on, off 혹은 error이다. 공장자동화모의체계를 위하여 필요한 본질적인 상태는 카메라가 켜지거나 꺼진다는것이다. 만일 켜지면 카메라는 화상을 얻을수 있으며 꺼지면 얻을수 없다. 이와 같이 비데오카메라의 추상화는 2개의 값을 가지는 상태속성을 가지는데 그 상태값은 on과 off이다. 카메라의 상태를 조종하여야 하며 그것을 위하여 카메라가 켜졌는가, 꺼졌는가 하는 통보들을 접수하여야 한다.

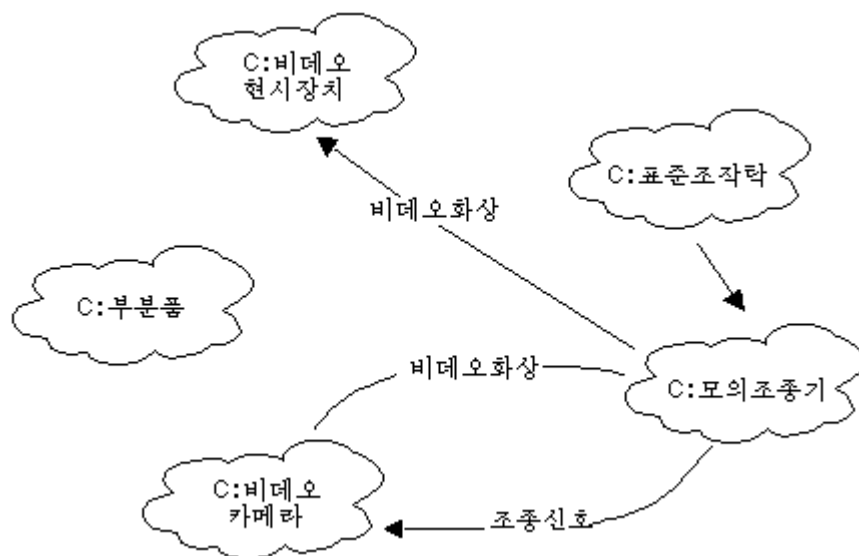
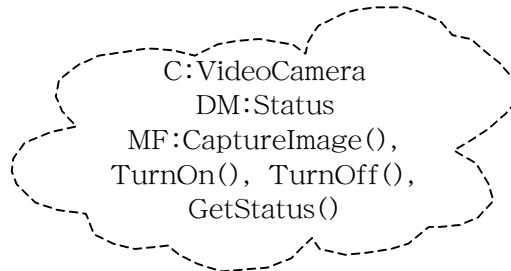



그림 7-13. 공장자동화를 위한 객체들의 호상작용

또한 카메라의 상태를 결정하여야 한다. 이 기능들은 카메라의 추가적인 동작들로 된다. 비데오카메

라의 추상화는 단일속성을 가지며 그것은 상태와 다음과 같은 동작을 가진다.

- 화상을 포착한다.
- 카메라를 켠다.
- 카메라를 끈다.
- 카메라의 상태를 얻는다.

그림표시법을 리용하면 클라스 VideoCamera는 아래와 같다.

목록 7-1의 C++클라스선언을 통하여 이러한 추상화를 실현할수 있다. 설계공정이 3단계를 가진다는데 주의하시오.

1. 체계에서 객체 혹은 클라스들을 결정한다.
2. 객체들이 어떻게 호상작용하는가를 결정한다.
3. 객체들의 동작과 속성을 결정한다.

목록 7-1. vcam.h에서 VideoCamera클라스선언

```
#ifndef VIDEOCAMERA_H
#define VIDEOCAMERA_H
#include " image.h"
enum CameraStatus {CameraOn, CameraOff };
class VideoCamera {
    public:
        VideoCamera();
        CameraStatus GetStatus() const;
        VideoImage CaptureImage();
        void TurnOn();
        void TurnOff();
    private:
        CameraStatus Status;
};
#endif
```

이것들은 임의의 객체지향설계방법론들에서 쓰이는 표준적인 단계들이다. 일반적인 설계처리들처럼 이 단계들은 설계가 완성될 때까지 반복된다. 례를 들어 객체의 동작을 결정할 때 이 객체와 다른것들과의 호상작용을 다시 고찰하여야 한다는것을 알수 있을것이다. 이와 함께 객체의 속성을 결정할 때 당신

은 그 속성을 표현하기 위하여 추가적인 객체들을 쓰게 된다는것을 알게 된다.

클라스 VideoCamera의 선언에서 성원함수 CaptureImage()는 VideoImage객체를 돌려 준다. 이 객체는 비데오화상객체의 추상화이다. 그것은 카메라에서 가정한 조종자에로 보내 주는 객체이다. 이 객체를 실현하기 위하여 OOA/OOD공정을 적용하여 보자. 비데오화상은 그것을 표현하는것이 명백하지 않기때문에 흥미 있는 객체라고 말할수 있다. 그림 7-13으로부터 비데오화상이 모의조종기에로 보내진다는것을 알수 있다. 그 모의조종기는 현시장치의 전에 있다. 현시장치가 부분품들의 모임을 현시하여야 하므로 명백히 비데오화상은 충분한 정보를 요구하며 이에 의하여 현시장치는 적당한 화상을 조합한다(즉 적절한 크기와 색갈을 가진 RectangleShape를). 결과로 도형객체 RectangleShape를 리용하면 이 객체에 표현이 유혹적으로 된다. 그러나 일반적으로는 설계착오를 가져 올수 있다.

객체 VideoImage를 실현하기 위하여 RectangleShape를 리용하면 VidoeImage를 현시하기 위하여 계획한 방법보다는 비데오화상의 추상화를 매우 빠르게 한다. 론리적인 오유를 설명하기 위하여 EzWindows와 같은 창문체계를 쓰지 않는 현시장치로 동작한다고 가정해 보자. 이 장치로 우리는 카메라에서 오는 부분품모임을 현시하는 특징문자렬모임을 내보내야 한다(즉 붉은 부분품과 푸른 부분품, 또 다른 붉은 부분품인 부분품모임에 대하여 Red Blue Red라는 문자렬을 내보낸다). 비데오화상이 RectangleShape에 의해 현시될 때 이러한 변화를 만들기 위하여 시간을 소비할것이며 예상한것보다 더 많은 코드를 변경할것을 요구한다. 일반적으로 정보가 현시되는 방법을 자주 바꿀것을 요구하기때문에 체계의 중지조작에서 정보의 현시를 분리하는것은 좋은 생각이다.

이러한 문제를 해결하기 위하여 다시 추상화를 적용하여 적합치 않은것은 소거하고 본질에 초점을 둔다. 공장자동화흐름선에서 비데오화상은 부분품들을 찍어 놓은것이다. 이렇게 비데오화상은 3개의 부분품으로 이루어 진다. 목록 7-2는 VideoImage에 대한 클라스선언이다.

목록 7-2. image.h에서 VideoImage클라스의 선언

```
//VideoImage클라스
#ifndef VIDEOIMAGE_H
#define VIDEOIMAGE_H
#include " part.h"
class VideoImage {
    public:
        VideoImage(const color &c1, const color &c2,
        const color &c3);
        Part GetPart1() const ;
        Part GetPart2() const;
        Part GetPart3() const;
    private:
        Part Part1;
        Part Part2;
        Part Part3;
};
#endif
```

VideoImage가 하나의 Part를 포함한다는데로부터 다음의것을 설계하자. Part는 리용하지 않는 VideoImage도 포함한다. 추상화에 의한 부분품의 본질적인 속성 혹은 상태는 그의 색갈이다. 보통 색갈속성에 맞는 검토자를 요구한다. 대응한 변이자를 요구하지 않는데 그것은 일단 하나의 부분품이 창조되면 그 색갈을 바꿀 필요가 없기때문이다. 목록 7-3은 Part에 대한 클라스선언이다.

목록 7-3. part.h에서 Part클라스의 선언

```
#ifndef PARH_H
#define PART_H
#include " ezwin.h"
class part {
    public:
        Part (const color &c = Red);
        color GetColor() const;
    private:
        color Color;
};
#endif
```

양성모의기에서 마지막 2개의 객체는 비데오현시장치와 조종탁이다. 이 객체들은 사용자대면부객체들이다. 조종자는 비데오현시장치의 부분품을 보게 되며 남자 혹은 녀자가 조종탁을 리용하여 응답을 진행한다. 비데오현시장치로부터 시작하자. 비데오현시장치의 임무는 화상을 설계하는것이며 그리하여 조종자는 그것을 보고 부분품모임을 접수하겠는가, 거절하겠는가를 결정한다. 비데오카메라처럼 비데오현시장치도 그것이 켜졌는가, 꺼졌는가 하는 상태정보를 가진다. 부분품들의 화상을 현시하기 위하여 Ezwindows서고로부터 SimpleWindow클라스를 리용한다. 현시를 위하여 리용되는 SimpleWindow클라스는 현시장치의 다른 속성이다. 이 수법은 비데오현시장치클라스에서 부분품들을 어떻게 현시하는가 하는 측면을 밀페시키려고 하므로 적합하다. 클라스 VideoDisplayUnit의 선언은 목록 7-4에 있다.

구축자는 string파라메터를 요구하는데 그것은 창조되는 SimpleWindow의 제목띠에 나타나는 문자렬이다. 성원함수 DisplayImage()는 VideoDisplayUnit가 부분품들의 화상을 현시하도록 하는 통보문이다. 통보문의 내용은 비데오화상이다.

목록 7-4. dunit.h에서 VideoDisplayUnit클라스의 선언

```
#ifndef DISPLAYUNIT_H
#define DISPLAYUNIT_H
#include <string>
#include " ezwin.h"
#include " image.h"
enum DisplayStatus { DisplayOn, DisplayOff };
class VideoDisplayUnit {
```

```
    public:
        VideoDisplayUnit(const string, &Title);
        void DisplayImage(const VideoImage &Image);
        void TurnOn();
        void TurnOff();
    private:
        SimpleWindow W;
        DisplayStatus Status;
};
#endif
```

설계에서 마지막객체는 조종탁이다. 조종탁의 임무는 양성생에게 임의의 명령을 현시하고 조작자양성생으로부터의 모든 입력량을 접수하는것이며 양성대화의 결과를 현시하는것이다. 오직 조종탁의 속성들만이 그의 상태 즉 켜기와 끄기이다. 목록 7-5는 클라스 Console의 선언을 보여 준다. 성원함수 TurnOn()은 조종탁을 On으로 설정하는데 그것은 창문을 재설정하는 조작자를 가리키는 머리글통보문을 인쇄하도록 한다.

GetResponse()는 응답을 읽으며 모의조종기들에 그것을 확정할수 있게 돌려 준다. 성원함수 PrintSessionResults()는 가정이 끝났을 때 조종탁우에서의 양성대화결과를 현시한다. 클라스정의를 리용하여 양성모의기의 조작을 설계할수 있다. 모의단계의 고급한 서술은 다음과 같다.

단계 1. 부분화상을 얻는다.
단계 2. 현시창문에 부분품화상을 현시한다.
단계 3. 양성생의 응답을 읽고 기록한다.
단계 4. 응답점수를 매긴다.
단계 5. 양성대화가 끝났는가, 안끝났는가를 검사한다. 만일 등록시간이 없다면 단계 1로 간다. 등록된 시간이 있다면 양성대화의 마지막통계표가 계산되고 인쇄된다.

이 단계들은 모의의 중요한 순환고리를 이루는것들이다.

목록 7-6은 양성모의기를 실행하는 코드를 보여 주고 있다. 코드의 첫 부분은 모의의 중심적인 객체들인 TrainingMonitor와 InspectionCamera, TrainingConsole를 창조한다. 구축한후에 직접 이 장치들을 켠다. 코드의 두번째 부분은 필요한 여러가지 계수기들을 초기화하고 모의의 시작시간을 기록한다. 코드의 중요한 부분은 이미 렬거한 완전히 일치하는 모의의 중요한 순환고리이다.

모의프로그람에 대한 다음의 4가지 리론은 주목할 가치가 있다. 첫째로 루틴프로그람 ApiMain()에서 대부분의 론리적인 프로그람을 보관하는것을 선택하였다. 하나의 모의조종기객체를 창조할수 있었으며 거기에 코드를 배치할수 있었다. 그러나 이 수법은 강제적인 수법이므로 그것을 고려하여 선택하였다.

목록 7-5.   console.h에서 Console클라스의 선언

```
#ifndef CONSOLE_H
#define CONSOLE_H
eunm ConsoleStatus { ConsoleOn, ConsoleOff };
```

```
class console {
    public:
        Console();
        void TurnOn();
        void TurnOff();
        char GetResponse();
        void PrintSessionResults(long Time, int Attempts,
            int Correct, int wrong);
    private:
        ConsoleStatus Status;
};
#endif
```

목록 7-6. trainer.cpp에서 공장자동화모의기의 실현

```
#include <iostream>
#include "uniform.h"
#include "dunit.h"
#include "ezwin.h"
#include "vcam.h"
#include "console.h"
using namespace std;
const long TestTime = 60 *1000L;
bool CheckResponse (char Response, const VideoImage &V) {
    Part Part1 = V.GetPart1();
    Part Part2 = V.GetPart2();
    Part Part3 = V.GetPart3();
    if (Part1.GetColor() = = Part2.GetColor()
      || Part1.GetColor() = =  Part3.GetColor()
      || Part2.GetColor()  = = Part3.GetColor())
        return Response = =  ‘a’ ;
    else
        return Response = =  ‘r’ ;
}
int ApiMain() {
    VideoDisplayUnit TrainingMonitor(" PARTS DISPLAY");
    VideoCamera InspectionCamera;
    Console TrainingConsole;
    TrainingMonitor.TurnOn();
    TrainingConsole.TurnOn();
```

```
        InspectionCamera.TurnOn();
        InitializeSeed();
        int Attempts = 0;
        int CorrectResponses = 0;
        int IncorrectResponses = 0;
        const long StartTime = GetMilliseconds();
        long ElapsedTime;
        do {
        VideoImage Image = InspectionCamera.CaptureImage();
        TrainingMonitor.DisplayImage(Image) ;
        char Response;
        Response = TrainingConsole.GetResponse();
        ++Attempts;
        if (CheckResponse(Response. Image))
            ++CorrectResponses;
        else
            ++IncorrectTrsponses;
        ElapsedTime = GetMilliseconds();
        } while ((ElapsedTime - StartTime) < TestTime);
        Trainingconcole.PrintSessionResults(TestTime,
            Attempts, CorrectResponses, IncorrectResponses);
        TrainingMonitor.turnOff();
        TrainingConsole.TurnOff();
        InspectionCamera.TurnOff();
        return 0;
    }
```

둘째로 프로그람의 심장부인 ApiMain()함수에서 do-while순환부는 매우 자연스럽다. 프로그람은 양성자의 물리적인 실현과 완전히 일치한다. 셋째로 이 프로그람은 유지와 확장이 쉽다. 례하면 다음장에서의 련습은 부분품모임을 접수할것인가, 거절할것인가를 조종자가 마우스를 리용하여 할수 있게 프로그람을 변경하는것이다. 객체지향설계를 하였으므로 이러한 변경을 하기가 쉬울것이다. 넷째로 개발한 객체들은 다른 모의실현에서 재리용될수 있다. 중앙고속도로운행감시소의 모의기를 실현한다고 가정하자. 공장양성모의기로 우리가 개발하였던 대부분의 코드는 재리용될수 있다. 고속도로운행감시체계에서 조작자는 조종실에 앉아 고속도로운행을 보여 주는 비데오화면들을 감시한다. 비데오카메라들은 고속도로상에서 중요한 위치에 자리 잡고 있다. 만일 운행에서 일부 장애가 생기면 조작자는 고속도로순찰에게 통지한다.이 체계를 실현하기 위하여 공장자동화양성기로 개발한 많은 객체들을 재리용할수 있다. VideoImage클라스를 재리용할수 있는데 부분품대신에 비데오화상은 승용차로서 구성된다. 클라스 VideoDisplayUnit와 VideoCamera들은 적게 혹은 전혀 수정되지 않고 재리용될것이다. 그것을 재리용하는 리유는 0으로부터 시작하여 그것을 만드는데 드는 시간의 절반만큼한 시간동안에 고속도로감시체계

모의를 설계하고 실현하고 시험할수 있을것이다. 셋째와 넷째에서 요점은 확장성과 재리용이 객체지향설계와 실현의 중요한 두가지 리득이라는것을 보여 준다.

7.7 알아 둘 점

✓ 실세계객체들을 표현하는 새로운 자료형들은 C++의 클라스구조를 통하여 창조된다.
✓ 클라스형객체는 2개의 부분모임인 속성모임과 동작모임을 가진다. 속성들을 자료성원, 동작들을 성원함수들이라고 말한다.
✓ 새로운 클라스형을 만들 때 그다음 중요한 두 초기단계는 객체의 속성과 동작을 결정하는것이다.
✓ 객체에 속성값을 돌려 주는 성원함수들을 검토자라고 한다.
✓ 객체의 속성값을 설정 혹은 바꾸는 성원함수들을 변이자라고 한다.
✓ 일부 기능 혹은 동작을 수행하는 객체들을 가리키는 성원함수들을 촉진자라고 한다.
✓ 구축자는 객체가 창조되는 경우 호출되는 특수한 성원함수이다. 그것은 일반적으로 객체의 속성들을 초기화하는데 리용된다.
✓ 클라스선언은 두 부분 즉 공개부(public)와 비공개부(private)로 나누어 진다. 공개부에는 객체사용자들에 의해 접근될수 있는 객체의 속성들과 동작들에 대한 선언이 들어 있다. 구축자는 항상 공개부에서 선언되여야 한다. 비공개부에는 객체사용자들에게 보이지 않거나 접근될수 없는 성원함수들과 자료성원들이 들어 있다. 이 성원함수들과 자료성원들은 오직 그 객체의 성원함수에 의해서만 접근될수 있다.
✓ 객체를 창조하는 처리를 구체례만들기라고 한다.
✓ 자료성원들과 성원함수들은 성원접근연산자 즉 점(.)을 리용하여 접근된다.
✓ 상수성원함수는 객체의 속성들을 변화시킬수 없는 함수이다.
✓ 기정인수들은 더 유연하고 편리하게 구축자를 만들어 준다.
✓ 참조자료성원들은 객체가 구체례화될 때 초기화되여야 한다. 일단 초기화된 참조자료성원들은 변경할수 없다.
✓ 추상화는 객체에 접합지 않은 속성들은 제거하고 본질적인것에 초점을 모으게 한다.
✓ 객체지향설계는 3개의 기초단계들로 구성되는데 첫 단계는 체계에서 객체들을 결정한다. 둘째 단계는 객체들이 어떻게 응답하는가, 혹은 호상작용하는가를 결정하며 세번째 단계는 객체들의 동작과 속성들을 결정한다.
✓ 훌륭한 설계는 견고하다. 체계를 설계하는데 시간을 보낸다고 걱정하지 말아야 한다. 실행이 길어지면 그만큼 시간은 절약된다.

콤퓨터의 력사

보관된 프로그람들

계산에 있어서 다음으로 중요한 발전은 콤퓨터가 프로그람화된것이다. ENIAC콤퓨터는 도선들이 접속될수 있도록 소케트들의 모임으로 구성된 접속기판에 의하여 프로그람화되였다. 여기서는 소케트들을 련결시키는 방법으로 기계회로를 변경시키고 각이한 계산들을 수행하였다. 문제는 이런 일이 대단히 많은 시간을 소비한다는것이다. 복잡한 문제들에 대해서는 기계를 다시 프로그람화하는데 여러 날이 걸리였다.

이 문제는 전자적으로 분리되여 있는 많은 콤퓨터(EDVAC)들의 개발에 의하여 해결되였다. 비록 EDVAC에 구현된 개념의 개발도상에서 일부 론쟁도 있었지만 대부분의 력사가들은 프로그람이 보관된 콤퓨터의 설계에서 폰 노이만이 중심적인 역할을 하였다는것을 인정하고 있다.

폰 노이만은 잘 알려 져 있으며 높은 존경을 받고 있는 훌륭한 수학자이다. 콤퓨터에 기여한것외에도 노이만은 경기론, 물리학, 리론수학, 기상학과 같은 부분에 중요한 기여를 하였다. 노이만은 ENIAC상에서의 작업을 고찰하기 위하여 펜실바니아종합대학을 찾은 후에 콤퓨터에 흥미를 가지게 되였다. 에커트와 모클리를 만난후에 노이만은 론문 《EDVAC에 관한 보고서초안》을 발표하였는데 그것은 모든 현대콤퓨터들의 구성요소들과 기본연산들을 개요한것이였다. 가장 중요한 개념들중의 하나는 콤퓨터를 조종하는 프로그람을 자료와 함께 기억기에 보관한것이였다. 이 개념은 프로그람을 자료처럼 조종할수 있으며 모든 콤퓨터들에 있어서 기초적인 기능으로 될수 있다는것을 의미하였다.

이 점에서 ENIAC와 EDVAC에 구현된 개념들이 발전도상에서 있은 일부 론의를 상기하고 언급할 가치가 있다. 펜실바니아종합대학의 많은 연구사들은 노이만의 론문에서 자기들의 이름을 생략하고 그 설계에 대한 자기들의 공적을 언급하지 않음으로써 자기들을 무시하였다고 생각하였다. 이런것과 지적소유권과 관련한 다른 갈등으로 하여 에커트와 모클리는 1946년에 대학을 떠나서 전자적인 콤퓨터를 설계하고 만들어 파는 회사를 설립하였다. 자기들의 발명을 보호하기 위하여 그들은 전자적인 콤퓨터의 조작을 담보하는 특허권을 얻었다.

ENIAC를 완성하기 3년전인 1942년에 아이오와주립대학교수인 죤 아타나소프, 대학졸업생인 클리포드 베리는 Atanasoff-Berry Computer 략칭 ABC를 만들었다. ABC도 역시 진공관을 리용하였으며 2진법을 사용하였다. 실제로 모클리는 아이오와주에서 아타나소프교수를 만나 전자적인 계산기를 만들데 대한 그의 제의를 검토하였다. 제의서의 대부분의 개념들은 ENIAC에서 수행하여야 할 길을 밝혀 주었다. 죤 아타나소프에게는 공교롭게도 그의 대리인이 ABC에서의 기술혁신을 담보하는 특허권을 받아 주지 않음으로써 응당한 명예를 실현할수 없었다.

ENIAC의 원리들에 대한 모클리-에커트특허권이 종국적으로 승인된 때인 1964년에야 비로소 전자적인 콤퓨터의 발전에 대한 모든 이야기가 충분하게 밝혀 지게 되였다. 이때 그 특허권은 스페리 랜드(Sperry Rand)에 의해 장악되였다. 1968년에 스페리는 특허권을 실시하고 특허권사용료를 받기로 결심하였다. 호니웰(Honeywell)회사는 특허권사용료를 지불하기보다는 법정에서 싸우기로 결심하였다. 호니웰의 전략은 쓸모 없게 된 ENIAC특허권을 가지는것이였다. 특허권은 그 특허권의 개념들이 이미 전에 개발된것이라고 인정될수 있을 때 무효로 선언될수 있다. 누구나 예측하듯이 죤 아타나소프는 소송기간에 기본증인으로 되였다. 5년간의 법정투쟁후에 재판정은 판사로부터 ENIAC특허권이 무효화 된 례를 듣고 《에커트와 모클리가 처음으로 전자적인 수자식자동콤퓨터를 발명하지는 못하였지만 죤 아타나소프박사로부터 기본주제를 이끌어 내였다.》라고 판결하였다. 25년이 지난후에야 전자적인 수자식콤퓨터를 개발하는데 기여한 죤 아타나소프의 공적이 공개되였다.

프로그람이 기억된 콤퓨터의 개념이 출현함으로써 정보보관체계들이 화제에 오르게 되였다. 이 부문에서는 영국의 연구사들이 전파탐지기로 일을 한것으로 하여 앞자리에 서게 되였으며 따라서 첫 조작프로그람저장기계를 만들어 냈다. 저장프로그람의 개념을 받아 들인 기계의 원형을 Manchester Mark1 이라고 하였는데 이것은 영국의 만체스터종합대학에서 1948 년에 완성되였다. 그것은 매개가 32 비트로 된 32 개 단어들의 기억기를 가지였다. 1 년후에 영국의 캠프리지에서 다른 프로그람이 보관된 콤퓨터 Electronic Delay Storage Automatic Computer(EDSAC 전자적인 지연보관의 자동콤퓨터)가 공개되였다. EDVAC 는 이 모든 개발을 산생시켰지만 그의 기초설계가 1952 년까지 완성되지 못하였다.

참고할 책

객체지향분석과 설계에 대한 여러가지 좋은 책들은 다음과 같다.

- G.Booch, object-Oriented Analysis and Design with Applictions, Redwood City, CA:Benjamin-Cummings, 1994
- R.Martin, Designing Object-Oriented C++ applications Using The Booch method, Englewood Cliffs, NJ: Prentice-Hall, 1995.
- J.Rumbaugh, M.Blaha, W.Premerlani, F.Fddy, and W.Lorensen, Object-Oriented Modeling and Design, Englewood Cliffs, NJ:Prentice-Hall, 1991.
- R.Wirfs-Brock, B.Wilkerson, and L.Wiener, Designing Object-Oriented Software, Englewood Cliffs, NJ:Prentice-Hall, 1990

련습문제

7.1 ______________은 클라스의 구체례를 창조하기 위하여 호출되는 함수이다.
7.2 ______________은 객체의 자료성원값을 얻는 성원함수이다.
7.3 ______________은 객체의 자료성원값을 바꾸거나 설정하는 성원함수이다.
7.4 ______________은 일부 동작이나 봉사를 수행하는 성원함수이다.
7.5 구조화설계는 전통적인 설계방법이다. 구조화설계에 대해 알아 보고 객체지향설계와 비교하시오.
7.6 다음과 같은 기하학적인 객체를 표현하는 C++클라스선언을 하시오.

1) 원
2) 4각형
3) 타원
4) 3각형
5) 평행4변형

7.7 다음과 같은 클라스선언이 있다.

```
class Top{
    public:
        Top();
        int Look();
    private:
        int Value;
        void Change(int v);
};
```

그리고 다음의 코드부분을 참고하시오.

```
int main()  {
    Top t;
    t.value = 3;          //옳은가, 틀리는가?
    int k = t.Look();     //옳은가, 틀리는가?
    t.Change(3);          //옳은가, 틀리는가?
    return 0;
```

```
    }
```

설명을 붙인 명령들이 옳은가 틀리는가를 지적하시오.

7.8 다음의 객체들을 표현하는 C++클라스를 선언하시오.

1) 수자식기계
2) 가정용온도조절기
3) 수자식시간조절기
4) 록음기

7.9 다음의 응용프로그람에 대하여 공장자동화양성자체계에서처럼 객체지향설계를 하시오.

1) 콤퓨터
2) 달력
3) 은행자동현금출납기
4) 서고검사체계
5) 자동판매기
6) 맞춤법검사기

설계를 통하여 객체들이 어떻게 련결되고 관계되는가를 알수 있다.

7.10 다른 통보문들이 RectangleShape에 리용될수 있다. 례를 들어 지우기통보문은 현시장치로부터 화상을 지워야 할 4각형을 가리킨다. 다른 유용한 통보문을 생각할수 있겠는가? 매 통보문들에 대한 성원함수들을 포함시켜 RectangleShape 클라스를 확장시키시오.

7.11 SimpleWindow안에서의 위치를 지정하는 파일을 읽는 프로그람을 작성하시오. 위치를 현시하는 류동소수점자료들은 공백으로 구분된다. 매 위치를 읽기 위하여 해당 위치에 너비 1cm, 높이 2cm인 붉은 4각형을 그린다. 창조한 SimpleWindow는 10×10cm이여야 한다. 프로그람은 유효한 자리표들을 읽고 현시하는 기능을 가져야 한다(즉 그것들이 창문안에 놓이게).

7.12 다음의 바른4각형객체들에 대한 선언을 리용하여 질문 1)-4)까지 대답하시오.

```
class Square {
      public:
         Square(SimpleWindow &W);
         void Draw();
         void SetColor(const color &color);
         void SetPosition(float Xcoord,
         float Ycoord);
         void SetSize(float Length);
      private:
          color Color;
         float XCenter;
         float YCenter;
         float SideLength;
         SimpleWindow &Window;
};
```

1) 창문 simple에 현시되는 GreenSquare라는 한변이 1.5cm인 바른4각형객체를 정의하고 그리는 명령렬을 작성하시오. GreenSquare의 중심위치는 창문의 왼쪽변두리로부터 3.5cm, 꼭대기에서 2.5cm 되는 곳에 있다. GreenSquare는 록색이여야 한다.

2) MagentaSquare라는 한변이 2cm인 바른4각형객체를 정의하고 그리는 코드토막을 작성하시오. MagentaSquare는 창문 Simple에 현시된다. MagentaSquare의 왼쪽웃구석의 위치는 창문의 왼쪽웃구석이다. MagentaSquare의 색은 당홍색이다.
3) 둘다 한변이 2.5cm인 두개의 바른4각형객체를 정의하고 그리는 코드를 작성하시오. 창문 DoubleSquare에 2개의 바른4각형을 현시한다. 하나의 바른4각형은 TopSquare라고 하며 다른 바른4각형은 ButtomSquare라고 한다. TopSquare의 위치는 창문의 왼쪽변두리로부터는 3cm, 꼭대기에서부터 1.5cm 되는 곳에 있다. ButtomSquare의 위치는 ButtomSquare의 웃변이 TopSquare의 아래변에 놓인다. TopSquare의 색은 붉은색이고 ButtomSquare의 색은 푸른색이다.
4) 3개의 바른4각형객체들을 정의하고 그리는 코드를 작성하시오. 하나의 바른4각형은 한변이 3cm이며 다른 4각형은 한변이 2cm, 마지막 4각형은 1cm이다. 창문 OverlappedSquare에 3개의 바른4각형들을 모두 현시하시오. 3개의 바른4각형의 위치는 모두 변수 XCenterWindow와 YCenterWindow에 포함되는 화면자리표들이다. 한변이 3cm인 바른4각형은 BigSquare이며 노란색이다. 한변이 2cm인 바른4각형은 MiddleSquare라고하며 록색이다. 한변이 1cm인 바른4각형은 SmallSquare라고 하며 붉은색이다.

7.13 Square에 대하여 다음의 선언이 리용된다고 가정하고 련습 7.12를 수정하시오.

```
class Square {
    public:
        Square(SimpleWindow &W, float Xcoord = 0,
          float Ycoord = 0, const color &c = Red,
          float Length = 1);
          void Draw();
          void SetColor(const color &Color);
          void SetPosition(float Xcoord,  float Ycoord);
          void SetSize(float Length);
    private:
          color Color;
          float XCenter;
          float YCenter;
          float SideLength;
          SimpleWindow &Window;
};
```

7.14 2차원상에서 구조물들의 공간관계를 직관적으로 보여 줄수 있는 프로그람을 작성해 달라고 요청받을수 있다. 구조물을 표현하려면 EzWindow형의 RectangleShape를 리용하여야 한다. 프로그람은 구조물현시에 리용하는 구조물의 차원과 위치, 색갈을 포함하는 파일을 읽어야 한다. 그 입력의 구체례는

width height xcoordinate ycoordinate color

인데 거기서 width와 height는 현시하려는 4각형의 크기(cm로)이고 xcoordinate과 ycoordinate는 구조물을 그리는 창문안에서의 위치이며 color는 그리는 4각형의 색이다. 모든 입력자료는 색갈을 제외하고는 모두 류동소수점자료이다. 입력자료색갈은 구조물의 색갈을 지정해 주는 문자렬이다. 가능한 값은 Red, Green, Blue, Yellow, Cyan 과 Magenta이다. 실례로 프로그람의 다음과 같은 자료파일을 읽으면 그림 7-14에서 보여 준것처럼 현시된다.

0.5 0.3 3.0 3.0 Red
0.3 0.2 5.0 5.0 Green
1.0 8.0 5.0 Blue
0.5 0.5 1.0 1.0 Magenta
0.5 1.0 2.0 6.0 Yellow

7.15 Kaleidoscope클라스를 설계하시오.

7.16 다음과 같은 대상에 대한 클라스를 선언하시오. 설계와 일치하게 C++클라스선언을 하시오.

1) Ezwindow도형체계에서 리용하는 화면위치
2) Ezwindow도형체계에서 리용하는 선객체
3) Ezwindow도형체계에서 리용하는 마우스객체

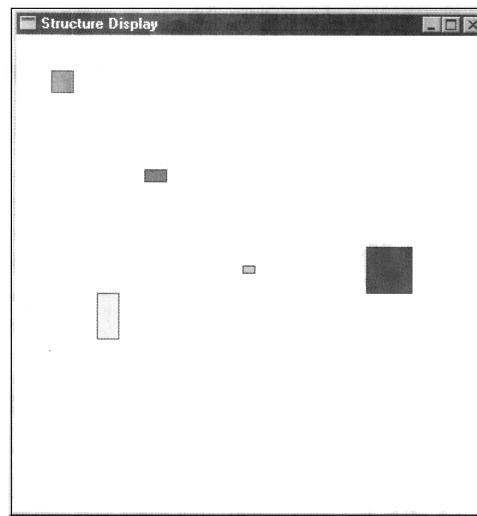



그림 7-14. 련습문제 7.14의 구조관계현시

7.17 부분품모임이 5개의 객체들로 구성된 공장자동화모의체계 VideoImage클라스를 다시 설계하시오. 수정하여야 할 다른 클라스는 무엇인가?

7.18 부분품이 폭과 높이를 가지도록 공장자동화모의체계의 Part클라스를 다시 설계하시오. 수정하여야 할 다른 클라스는 무엇인가?

7.19 매 장식품의 색갈이 우연적으로 주어 지도록 Kaleidoscope프로그람을 수정하시오.

7.20 화면을 4개의 구역으로 나누는 축우에 장식품을 그릴수 있도록 Kaleidoscope프로그람을 수정하시오(그림 7-15를 보시오). 프로그람은 장식품을 대각선상으로 배치하겠는가, 수직축상에 배치하겠는가는 임의로 결정하여야 한다.

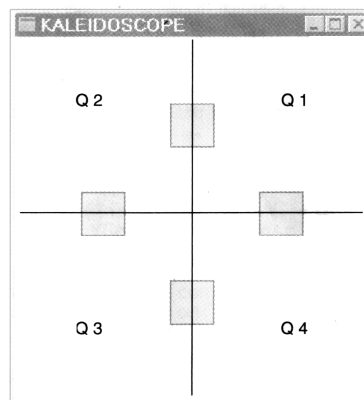



그림 7-15. 수평축과 수직축의 장식품위치

제 8 장. 추상자료형의 실현

소 개

자료추상화란 정보와 그 정보에 관하여 수행되는 연산들의 표현이다. 추상자료형(ADT)은 정보은페원리를 리용하는 잘 정의되고 완벽한 자료추상화이다. ADT들은 일반적으로 객체들을 창조하고 조종하도록 해준다. 이 장에서는 C++에 속해 있는 클라스들과 함수들 그리고 연산자들가운데서 ADT들을 소개하려고 한다. 유리수, 모조란수, 간단한 추측유희에 대한 ADT들을 개발하는 과정을 소개하려고 한다. ADT들을 개발하는 과정에 그의 대면부와 실현부에 대해서도 론의한다.

기본개념

- 자료추상화
- 추상자료형(ADT)
- 최소화규칙
- 클라스최소화원리
- 기정구축자
- 복사구축자
- 성원값주기
- 검토자
- 변이자
- 촉진자
- const 성원함수
- 해체자
- 보조함수와 연산자
- 연산자다중정의
- 다중정의된 삽입연산자와 추출연산자
- 참조귀환
- 모조란수

8.1 추상자료형의 소개

보다 복잡한 문제에 대한 해결책은 정보들을 표시하는것이다. 객체지향프로그람작성기술에서 그런식으로 수행되게 될 식과 연산은 하나의 자료추상화를 형성하게 된다. 이 장에서 유리수, 란수, 간단한 추측유희들에 대한 추상화를 진행하게 된다. 그 추상화는 클라스와 함수 그리고 연산자들을 리용하여 진행된다.

추상화를 개발하는데서 정보은페화원리는 무조건 동반되게 된다. 실례에서와 같이 정보은페화를 실현하면 자료의 완전성을 유지하는데 도움을 준다(례를 들면 분모가 0인 유리수를 피한다). 공개된 방법들을 리용하면 사용자프로그람들은 추상화가 완성되였으므로 변경에서 영향을 받지 않는다.

잘 정의된 추상화는 객체들이 직관적인 방식으로 창조되고 리용되도록 한다. 그러므로 추상화객체들의 정의와 조종을 위한 프로그람작성문법은 비교동작을 수행할수 있는 기본형과 표준클라스객체들과 형을 가진다. 례하면 유리수들에 대한 추상화 Rational을 가지고 1/2과 1/3을 더한 결과를 표시하려고 한다고 하자.

```
Rational a(1, 2); //a=1/2
Rational b(2, 3); //b=2/3
```

```
cout << a << " + " << b << " = " << (a + b) << end1;
```

이 코드는 두개의 int 혹은 float객체들의 합을 표시하는것과 같은 형태를 가진다. 이러한 형태는 전통적인 수법이 아니므로 C와 같은 객체지향이 아닌 언어로도 수행할수 있다. 전통적인 언어로서 새로운 객체들의 형태로 작업하려고 현재연산자들을 확장시킬수 있다. 프로그람작성자들은 함수들과 추가적인 형객체들을 정의하여야 한다. 결과 코드는 자연스럽지 못하고 조작하기가 힘들다.

정보은폐화원리를 리용하여 서고함수들은 추상적인 자료형 혹은 ADT와 결합된 잘 정의된 클라스를 호출하여야 한다.

8.2 Rational ADT기초

ADT의 론의를 유리수를 표현하는 ADT Rational의 개발로부터 시작한다. 유리수는 두 옹근수들의 비률로서 표준적으로 형식 a/b로 표시된다. a는 분자, b는 분모이다. 분모는 령이 될수 없다. 기초적인 산수연산은 다음과 같이 정의한다.

경험

왜 ADT에 대해 론의하는가?

여기서 ADT를 강조하고 있는것은 쏘프트웨어기사들이 거기에 중요성을 부여하고 있기때문이다. 문제해결과정을 설계할 때 쏘프트웨어기사는 그 응용프로그람에 대한 필요한 ADT들을 개발하는데 큰 관심을 돌리는데 그것은 ADT들이 객체지향프로그람작성의 기초이기때문이다.

실험결과는 쏘프트웨어개발성과가 객체지향프로그람작성을 리용하지 않은것보다 그것을 리용한것이 더 성공적이라는것을 말하여 주고 있다. 이 성과의 비결은 객체지향프로그람개발이 수속지향프로그람개발보다 더 간단하기때문이다. 실례로 성원함수들은 일반적으로 비성원함수들의 파라메터목록보다 더 작은 파라메터목록을 가지는데 그것은 거기에서는 객체의 자료성원들을 넘겨 줄 필요가 없기 때문이다.

잘 작성된 성원함수들은 자료성원들이 정확히 초기화되고 갱신될수 있게 하며 다른 프로그람함수들은 어떠한 적합한 객체값들을 리용해야 하는것을 확인할 필요가 없다. 또한 교갑화에 의해 제공된 접근보호가 의뢰기들이 공개대면부를 리용하여 집행이 규정시간외에 수행될 때 예측할수 없는 서고집행의 오유를 막아 주는것을 담보한다.

- 더하기: $\frac{a}{b}+\frac{c}{d}=\frac{ad+bc}{bd}$
- 덜기: $\frac{a}{b}-\frac{c}{d}=\frac{ad-bc}{bd}$
- 곱하기: $\frac{a}{b}\times\frac{c}{d}=\frac{ad}{bd}$
- 나누기: $\frac{a/b}{c/d}=\frac{ad}{bc}$

Rational을 개발하는 목적은 기본형들을 리용하여 정의된 객체로서 리용될수 있게 객체의 형을 자연적인것으로 만들기 위해서이다.

유리수를 표현하자면 분자와 분모를 표현하여야 한다. 이것은 유리수를 표현하는 클라스가 2개의 자료성원 즉 객체들의 한 성원은 특별히 분자를, 다른 성원은 분모를 표현하는 객체들을 정의하여야 한다는것을 말하여 준다. 두 자료성원은 모두 int객체들이다. 유리수들에 대한 선행경험은 Rational ADT의

성원함수들이 다음과 같은 방식으로 유리수객체들을 초기화하고 조종하는 방법을 제공한다는것을 말해주고 있다.

- 기정 혹은 개별적속성을 가진 유리수를 구축한다.
- 유리수에 다른 유리수를 더하고 덜고 곱하고 나눈다.
- 다른 유리수에 해당유리수의 값을 복사한다.
- 유리수와 다른 유리수를 비교한다.
- 유리수값을 표시한다.
- 유리수값을 얻는다.

의뢰기프로그람작성자의 활동과 정보은페를 지원하기 위하여 다음의 방법들도 유리수객체를 위하여 제공된다.

- 분자와 분모의 값을 검사한다.
- 분자와 분모의 값을 설정한다.

Rational클라스에 보충하기 위하여 또한 클라스서고에 일부 보조연산자들을 정의할 필요가 있다. 보조연산자(auxiliary operator)는 클라스성원이 아니라 산수연산자와 관계연산자, 흐름연산자의 다중정의판본이다. 이 연산자들은 매우 중요한데 그것들은 Rational객체들이 오유가 없이 자연스럽게 조종되도록 한다. 클라스와 함께 보조연산자들은 유리수서고를 구성한다. 이 연산자들을 Rational클라스의 성원이라기보다 보조적인것이라고 말하는 의미는 다음절에서 언급하기로 한다. 우의 코드에서 객체 a와 b의 정의를 그림 8-1에 주었다. 그림은 기억기가 분자와 분모를 표현하기 위하여 확보해 둔 구획을 가지는 유리수객체들과 결합된다는것을 보여 준다.

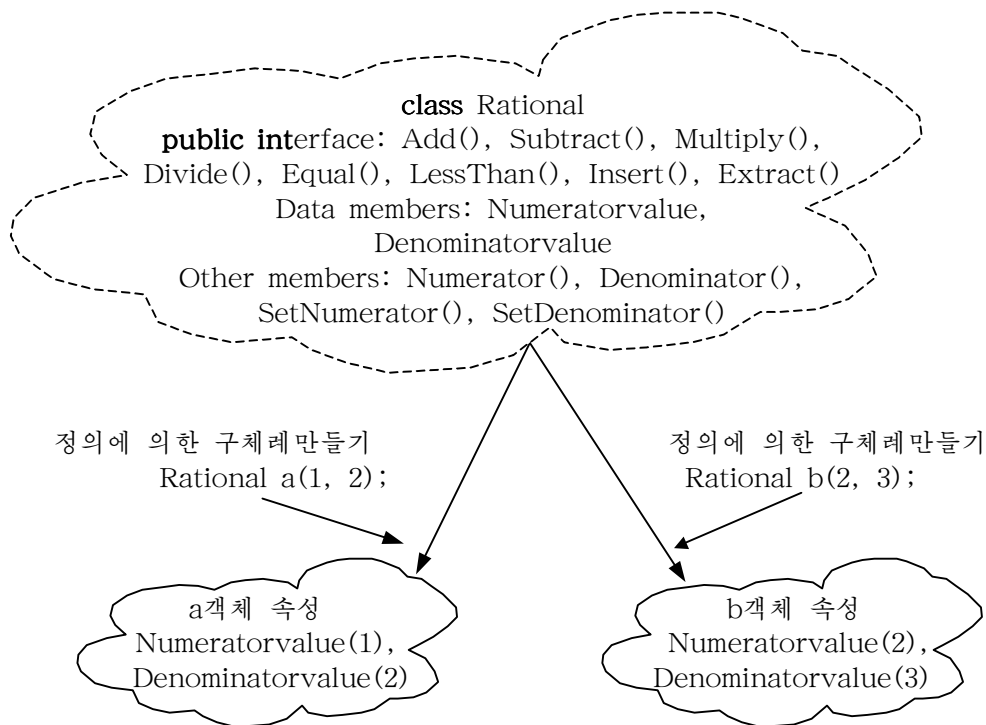



그림 8-1. 클라스정의에 의한 두 Rational객체의 구체례만들기

기억기의 다른 부분은 여러가지 성원함수들과 결합된 코드를 위하여 확보한것이다.

경험

최소화규칙

ADT서고안에 포함되는 동작을 결정하는것이 최소화규칙이다. 이 규칙은 만일 동작이 요구되지 않으면 그것은 ADT부분이 아니라는것이다. 이 규칙에는 여러가지 리유가 있다. 쏘프트웨어기술자들은 의뢰기응용프로그람개발자들이 최소크기의 서고를 좋아 한다는것을 알게 되였다. 이 방법으로 하면 서고를 포함하는 의뢰기응용프로그람이 가능한껏 작아 지며 넓은 범위의 기계구성우에서 실행될수 있다. 그리고 서고를 변경시켜야 하는 경우에도 보다 적은 원천코드를 고찰하면 된다. 최소화를 쓰면 프로그람효과를 높일수 있다.

최소화규칙에 대한 결론은 이제 서술하게 되는 클라스최소화원리이다. 만일 함수 혹은 연산자를 클라스의 성원이 되지 않게 정의할수 있다면 성원으로 쓰지 않는다. 이러한 규칙을 실현하면 클라스의 완성에 의한 변화에 대하여 비성원함수 혹은 연산자들이 일반적으로 영향을 받지 않게 한다.

8.2.1 유리수서고를 리용하는 의뢰기프로그람

유리수 ADT의 C++실현부를 보여 주기에 앞서 프로그람 8-1을 참고한다. 이 의뢰기프로그람은 우리가 Rational ADT의 여러가지 부분에 대하여 알고 싶었던것들에 대해 간단히 소개한다.

```
#include <iostream>
#include <string>
#include " rational.h"
using namespace std;
int main() {
    Rational r;
    Rational s;
    cout << "Enter rational number (a/b): ";
    cin >> r;
    cout << "Enter rational number (a/b): ";
    cin >> s;
    Rational t( r );
    Rational Sum = r + s;
    Rational Product = r * s;
    cout << r << " + " << s << " = " << Sum << end1";
    cout << r << " + " << s << " = " << Product << end1";
    return 0;
}
```

프로그람 8-1. Rational ADT의 기능을 보여 주는 프로그람

만일 유리수서고가 추상적인 자료형을 정확히 표시할수 있다면 프로그람을 쉽게 리해할수 있다.

의뢰기프로그람에서 두 입력자료가 유리수 1/2과 1/3이라고 가정하면 프로그람의 입력/출력동작은 다음과 같이 된다.

Enter rational number (a/b): <u>1/2</u>
Enter rational number (a/b): <u>1/3</u>
1/2 + 1/3 = 5/6

```
1/2 * 1/3 = 1/6
```

프로그람 8-1은 2개의 표준입출력흐름서고에 대한 머리부파일과 비표준유리수서고를 포함하는것으로부터 시작한다. 머리부파일 rational.h는 ADT들에 대한 대면부명세인 파일의 머리부파일 iostream과 류사하다(Rational의 실현부와 보조연산자들은 파일 rational.cpp에 있다). 약속한것처럼 머리부파일확장자 .h는 include명령문에 의해서 전체적인 이름공간에 서고대면부를 추가한다.

프로그람 8-1의 함수 main()은 처음 Rational객체 r와 s를 정의한다.

```
Rational r;
Rational s;
```

초기화는 객체들에 대하여 진행되지 않았다. 그것들의 초기값은 클라스 Rational에 관한 기정구축자에 의해 결정된다.

기정구축자는 파라메터를 가지지 않으며 객체의 여러가지 자료성원들에 기정값들을 설정한다. 구축자가 모든 자료성원들을 일부 조절한 값으로 초기화하는것은 표준적으로 실현된것이다. 0/1로 유리수를 기정적으로 표현하였다.

클라스는 일반적으로 여러개의 서로 다른 구축자들을 가질수 있다. 그 구축자가 객체정의에서 주어진 실제적인 초기화파라메터와 일치하는 파라메터목록을 가지지 않는다면 오유통보문이 표시된다.

항상 존재하는 구축자는 복사구축자이다. 복사구축자는 새로운 객체로서 같은 형태의 변하지 않는 참조파라메터를 가진다. 그 새로운 객체는 항상 존재하게 되는데 그것은 클라스설계자가 복사구축자를 지정하지 않아도 자동적으로 콤파일러에 의해 지원되기때문이다. 프로그람 8-1에서 복사구축자는 r를 t에 복사하여 만든다.

```
Rational t(r)    // t 는 r 의 복사이다.
```

항상 존재하는 클라스의 다른 성원은 기성의 객체를 변경시키는 값주기연산자(설계자가 그것을 쓰는가 안쓰는가에 무관계하다.)이다.

콤파일러에 의해 제공되는 복사구축자와 값주기연산자는 목적객체에 원천객체의 자료성원들을 성원별복사하는 기능을 수행한다. 성원별복사(memberwise copy)에서 여러가지 자료성원들은 원천자료성원으로부터 대응하는 목적자료성원에로 매 성원들이 비트방식(bit-by-bit manner)으로 직접 복사된다. 비트값주기(bit-by-bit assignment)란 기본형들을 리용하여 정의되는 객체우에서 수행되는 값주기이다. 자료성원들의 성원별복사가 이 장에서 우리가 참조하는 ADT들에 적합한것이므로 완성의 기초에서 복사구축자와 성원값주기연산자를 지원하는 콤파일러를 리용하게 된다. 그러나 8.5에서 명백한 정의가 요구된다면 성원들과 같은것을 정의하는 방법을 보여주게 된다.

객체 t의 정의로부터 형태적인 차이를 주시하면서 복사구축자는 객체 Sum과 Product를 정의하는데 프로그람 8-1이 리용되였다.

```
Rational Sum = r + s;
Rational Product = r * s;
```

값주기연산자(=)가 붙은 초기화표시가 있는 정의에서 호출된 구축자는 그 정의에서의 표시가 같은 하나의 파라메터만을 인용하는 구축자이다. 호출된 구축자는 실제파라메터로서 초기화표시를 리용한다. 그래서 Sum과 Product의 정의들은 둘다 복사구축자를 호출하는데 그것은 Rational더하기와 곱하기에 의해 생성되는 값들의 형태가 Rational이기때문이다.

다음의 코드부분에서 Rational객체 V는 Rational객체 W와 X에 복사된다. 객체 V는 값주기연산자를 리용하여 w에 복사되며 복사구축자를 리용하여 X에 복사된다.

```
Rational v(6, 21)  // v는 6/21
Rational w;        //w는 0/1
w=v                //w는  6/21로 값주기
Rational x=v;      //w는 복사구축자에 의해 6/21
```

W가 초기에 기정구축자를 리용하였으므로 0/1이라는것을 주의하시오. 객체는 다음의 값주기에 의해 식 6/21로 변경된다. 복사구축자보다 값주기연산자가 w에 대하여 더 리용되는데 그 리유는 w 가 변경되지만 정의되지 않기때문이다. X를 포함하는 명령은 값주기가 아니라 구축이 수행된다. 클라스는 또한 요구되는 클라스객체들을 초기화할수 있는 여러가지 정보를 가진 다른 구축자들을 정의할수 있다. Rational ADT에 대한 다른 구축자가 있을수 있다. 이러한 구축자는 하나 또는 2개의 옹근수파라메터를 요구한다. 첫 파라메터는 새로운 객체분자의 초기값이다. 두번째 파라메터는 기정값이 1이며 새로운 객체분모의 초기값을 지적하는 선택가능한 파라메터이다. 이 구축자의 실례를 다음의 코드부분에서 주었다.

```
Rational x(3, 4);  // x=3/4
Rational y(5);     // y=5/1
```

변이자 **const**는 **class**객체를 창조하는데 사용할수 있다. 변이자를 리용할 때 객체를 구축한후 그의 자료성원들의 값을 수정할수 없다 . 다음의 명령으로써 1/2를 표현하는 **const** Rational객체가 정의된다.

```
const Rational OneHalf(1, 2);
```

프로그람 8-1은 객체 r와 s에 값들을 할당하기 위하여 cin으로 2개 값을 얻는다.

```
cout << "Enter rational number(a/b): " ;
cin >> r;
cout << "Enter rational number(a/b): ";
cin >> s;
```

유리수추출연산자 >>는 Rational클라스의 성원연산자가 아니라 보조연산자이다. Rational추출은 성원메쏘드가 아니다. 그 리유는 하나의 추출을 얻기 위하여 특수한 형태가 요구되기때문이다. 추출연산자의 왼쪽연산수가 원천흐름이며 오른쪽연산수가 목적객체라는것을 알수 있다. 만일 연산자 >>가 Rational 클라스의 성원연산자로 다중정의된다면 두 연산수들은 성원연산자들에 대하여 거꾸로 될수 있다(메쏘드를 호출하는것이 왼쪽연산자이다). 이러한 뒤집기는 형태

```
r >> cin;
```

으로 얻을수 있는데 그것은 코드읽기와 헛갈릴수 있으며 무효로 될수도 있다. 추출연산자와 류사하게 삽입연산자는 유리수서고에서 보조연산자들로 정의되여 있다. 보조연산자의 왼쪽과 오른쪽연산수를 보조연산자로 만들기 위하여 정의하는것을 피할수 있다. 이미 본것처럼 프로그람 8-1은 Rational객체 Sum과 Product의 정의들도 포함하고 있다. 그 정의에서 리용된 표현들은 +와 *가 역시 다중정의되였다는것을 알수 있게 한다. 흐름연산자와 같이 산수연산자들도 성원연산자보다 보조연산자로 되는 편이 더 낫다. 산수연산자들을 성원연산자로 하기보다 보조연산자로 하는것이 더 나은 리유는 견고성이 더 강하기때문이다.

3장에서는 혼합된 기본형의 연산수가 존재할 때 식평가를 하는 동안 형변환(promotion)이 어떻게 자동적으로 시도되는가를 보여 주었다. 만일 혼합형연산수들이 클라스형객체들을 포함한다면 참(True)이다. 례를 들면 다음의 한 부분에서 더하기의 오른쪽연산수와 덜기의 왼쪽연산수는 Rational객체로 촉진이 진행된다.

```
Rational u(3, 4);
cout << (u + 2) << endl;
cout << (2 - u) << endl;
```

형변환은 연산자와 연산수의 적절한 결합에 대한 정의가 없을 때 진행된다. 형변환은 연산수에 대한 구축자를 호출하게 한다. 이전 실례에서처럼 2개의 기정파라메터를 가진 Rational구축자는 **int** 2라는 Rational을 구축할수 있다.그러므로 앞실례는 다음의 코드와 같다.

```
Rational u(3, 4);
Rational Two(2, 1);
cout << (u+ Two) << end1;
cout << (Two-u) << endl;
```

완성된 형변환규칙들이 많이 쓰인다. 이 규칙들의 형태적인 정의는 C++참조안내서를 통하여 보시오. 그러나 그것들을 리해하는데서 중요한 내용은 성원연산자로 취급될 때는 오직 오른쪽연산수만이 형변환될수 있으며 보조연산자로 취급될 때 임의의 연산수가 형변환될수 있다는것이다. 명령

```
cout << (u + 2) << end1;
```

에서 더하기연산자는 성원이나 보조연산자에 관계하지 않는다. 반면에 명령문

```
cout << (2 + u) << end1;
```

은 더하기연산자가 보조연산자일 때만 수행된다. 그리하여 일치성과 참조를 간단히 하기 위해서 산수연산자들을 보조연산자로 만든다.이런것은 Rational클라스객체들과 요구하는 동작들을 전부 볼수 있게 한다. 유리수서고의 실제적인 진행을 시작하자. 고찰하게 되는 유리수산수연산들은 더하기와 곱하기뿐이다. 다른 유리수산수연산들 즉 유리수의 관계연산들은 련습에 주었다.

유리수서고는 2개의 파일 즉 머리부파일 rational.h와 실현부파일 rational.cpp를 가지고 있다. 머리부파일의 검사부터 시작한다.

8.3 유리수대면부서술

머리부파일 rational.h를 복사한것을 목록 8-1에 주었다. 이 머리부파일은 엄격히 대면부서술이지만 이러한 속성이 모든 클라스머리부파일에 필수적인것은 아니다. 례를 들면 iostream과 같은 머리부파일은 전체적인 객체들을(즉 cin과 cout)정의한다.

rational.h에서 2개의 중요한 선언부분에는 전처리기명령문들이 포함된다. 첫 부분은 Rational클라스의 정의이다. 그뒤에 보조연산자들의 원형이 선언된다. 머리부파일에는 성원함수의 정의도, 보조연산자의 정의도 들어 있지 않다. 이 함수들과 연산자들의 정의는 실현부파일 rational.cpp에 주어 진다.

목록 8-1에서 처음 2개의 선언부분은 류사한 형태의 전처리명령문이다. 머리부파일에서 마지막행과 련결되는 첫 2개의 전처리명령은 파일이 콤파일장치안에 오직 한번 포함된다는것을 확인시킨다. 머리부

파일에서 다른 전처리명령은 iostream서고에 포함된다. 그 iostream서고는 쓰기와 추출연산자들의 보조적인 다중정의로서 포함된다.

8.3.1 접근제한

7장에서 만화경프로그람의 RectangleShape정의에서 보여 준것처럼 클라스선언이나 정의는 언제나 예약어 class로 시작한다. 그다음 클라스이름이 붙는다. 만일 클라스이름만 소개되여 있다면 그 선언에는 즉시 반두점이 뒤따른다. 만일 그것이 정의라면(즉 자료성원들과 성원함수들과 연산자들도 선언되여 있다면) 그 이름뒤에는 선언과 정의들의 렬이 놓인다.

목록 8-1. rational.h 머리부파일

```
//rational.h: Rational ADT의 선언
#ifndef RATIONAL_H
#define RATIONAL_H
#include <iostream>
#include <string>
using namespace std;
//Rational ADT:클라스 서술
class Rational {
    public:  //성원함수
        //기정구축자
        Rational ();
        // 두번째 구축자
        Rational(int numer, int senom = 1);
        //산수 및 흐름촉진자
        Rational Add(const Rational &r) const;
        Rational Multiply(const Rational &r) const;
        void Insert(osteam &sout) const;
        void Extract(istream &sin);
    protected:
        //검토자
        int GetNumerator() const;
        int GetDenominator() const;
        //변이자
        void SetNumerator(int numer);
        void SetDenominator(int denom);
    private:
        //자료성원
        int Numeratorvalue;
        int Denominatorvalue;
```

```
};
//Rational ADT:보조연산자서술
Rational operator+(const Rational &r, const Rational &s);
Rational operator*( const Rational &r, const Rational &s);
ostream& operator<<(ostream &sout, const Rational &s);
istream& operator>>(istream &sin, Rational &r);
#endif
```

그 렬은 왼쪽, 오른쪽대괄호로 분리된다. 접근지정자(access-specifier)표식을 리용하여 그 렬이 각이한 접근허가권들을 가진 부분으로 나누어 진다.

목록 8-1에서 Rational클라스정의는 세 부분을 가진다. 그것들은 접근지정자표식들인 **public**, **protected**, **private**로 련속적으로 시작된다. **public**안에서 선언된 성원들은 의뢰기프로그람에서 직접 리용될수 있다. 그 리유로 하여 프로그람 8-1은 여러개의 Rational구축자들을 리용할수 있게 된다. 클라스정의의 **protected**부분안에서 선언된 성원들은 표준적으로 클라스의 다른 성원함수들과 연산자들 그리고 파생클라스로부터의 성원함수들과 연산자들에 의해서만 리용될수 있다(즉 istream은 ios로부터 파생되고 istream객체는 그의 ios성원으로 호출된다). **private**부분안에서 정의된 성원들은 표준적으로 클라스의 다른 성원함수들과 연산자들에 의해서만 리용될수 있다.

8.3.2 구축자성원함수

목록 8-1에서 2개의 Rational구축자들의 기본형태는 형태적으로 함수기본형태와 다르다.

```
Rational ();
Rational(int numer, int denom = 1);
```

7장에서 클라스 RectangleShape에 대해 보면 구축자기본형은 귀환값을 지적하지 않는다. 클라스에 관한 구축자함수는 클라스와 같은 이름을 가질것을 요구하기때문에 다른 특별한 문법으로 구축자함수를 확인할 필요는 없다.

선언되여 있는 첫 구축자는 파라메터를 리용하지 않으므로 기정구축자이다. 두번째 구축자는 하나 혹은 2개의 객체들을 파라메터로서 요구한다. 2개의 파라메터이름들이 암시하는것처럼 그것들도 련속적으로 새로운 Rational객체의 분자와 분모자료성원들을 설정하여 준다.

분모에 대한 기정값도 1이다. 클라스선언에서 복사구축자도 아니고 성원대입연산자도 아닌것이 선언된것처럼 Rational클라스를 콤파일러에 의해 제공된 정의를 리용하게 된다.

8.3.3 촉진자성원함수

객체의 자료성원이나 성원함수를 참조하기 위하여 의뢰기코드는 연산자 (.)를 리용한다. 접근연산자의 왼쪽연산수는 객체이며 오른쪽연산수는 참조되게 되는 객체의 특수한 성원이다. 이미 string과 RectangleShape객체과 같이 접근연산자를 리용하였다. 례를 들면 다음과 같은 코드부분은 얻어 진 문자렬의 길이를 표시한다.

```
cout << "Enter a string: ";
string s;
```

```
cin >> s;
cout << "Length of " << s << " is " << s.size();
```

다음의 실례는 여러가지 Rational성원함수들을 지적하기 위하여 접근연산자를 리용한다.

```
Rational r;
Rational s;
r.Extract(cin);
s.Extract(cin);
Rational t= r.Add(s);
t.Insert(cout);
```

비록 접근연산자가 성원을 호출하기 위한 구조를 제공한다 해도 그것은 역시 유효한 문법에서 리용되여야 한다는것이다. 우의 코드부분에서 호출된 Rational성원들은 모두 클라스선언의 **public**부분에서 선언된다. 그래서 그것들은 의뢰기와 성원코드량쪽에서 다 호출될수 있다.

정보은페화원리가 일반적으로 뒤따르기때문에 자료성원들의 대부분은 **public**성원이 아니다. 다음으로 자료성원들도 표준적으로 비성원함수와 연산자들에서 접근연산자로 참조될수 있다. 비공개호출이 자료의 안전성을 담보하는데 도움을 주는 원인으로 된다는것은 옳다.

접근제한이 없는 객체의 명확치 않은 재리용은 불일치 혹은 비법적인 값들을 표현할수 없다. 그러므로 자료가 구축된 **public**대면부는 자료성원들을 안전하게 조종하기 위한 성원함수들과 연산자들도 지원한다. 그것들의 대면부는 정보은페화를 지원하여 준다. 이 성원함수들도 세 부류로 가를수 있는데 검토자, 변이자 그리고 촉진자이다. 검토자함수는 객체의 자료성원들의 표현에 접근하는 방법을 제공한다. 변이자함수는 객체의 자료성원들의 표현을 변경시키는 방법을 제공한다. 촉진자함수는 객체로 진행하려고 하는 연산들을 실현하기 위한 방법을 제공한다.

구축자선언후에 목록 8-1은 4개의 대수적인 흐름촉진자함수의 모임을 보여 준다(관계촉진자와 다른 산수촉진자들을 련습에 준다).

```
//산수와 흐름촉진자들
Rational Add(const Rational &r) const;
Rational Multiply(const Rational &r) const;
void Insert(ostream &sout ) const;
void Extract(istream &sin);
```

2개의 산수촉진자 Add()와 Multiply()는 하나의 파라메터 r를 가진다. 촉진자는 객체의 값을 호출하는데 리용되며 실제파라메터의 값은 성원함수의 결과를 결정한다.

흐름촉진자 Insert()는 파라메터로서 ostream에로의 참조를 요구한다. 그 촉진자는 흐름에 유리수를 삽입한다. 흐름촉진자 Extract()는 파라메터로서 iostream의 참조를 요구한다. Extract()는 변이자이다. 2개의 흐름촉진자들은 삽입 또는 추출동작이 흐름을 변화시키므로 참조파라메터를 가진다.

Rational촉진자들이 공개성원이므로 그것들도 사용자프로그람에 의해 리용될수 있다. 그러나 그것들은 실천에서 리용되지 않는다. 그것들은 자연적인 대면부를 가진 보조연산자대신에 리용된다. Rational촉진자들은 보조적인 연산과 흐름연산자들이 자기 일을 하도록 한다. 다음의 코드부분에서는 4개의 촉진자가 리용되였다.

```
Rational r;
Rational s;
cout << "Enter rational number (a/b): ";
r.Extract(cin);
cout << "Enter rational number (a/b): ";
s.Extract(cin);
Rational t(r);
Rational Sum = r.Add(s);
Rational Product = r.Multiply(s);
r.Insert(cout);
cout << " + ";
s.Insert(cout);
cout << " = ";
Sum.Insert(cout);
cout << end1;
r.Insert(cout);
cout << " * ";
s.Insert(cout);
cout << " = ";
Product.Insert(cout);
cout << end1;
```

이 코드토막은 코드작성을 유연하게 하는데서 보조연산자들이 얼마나 중요한 역할을 하는가를 보여준다. 프로그람 8-1에서의 명령들은 촉진자를 리용하는것이 더 유리하다는것을 보여 준다. Rational클라스선언에서 촉진자 Add(), Multiply() 그리고 Insert()의 선언뒤에 수식자 **const**를 리용하였다.

```
Rational Add(const Rational &r) const;
Rational Multiply (const Rational &r) const;
void Insert(ostream &sout) const;
```

파라메터목록뒤에 **const**예약어를 붙이면 객체의 값을 변경하지 않는 읽기전용함수로 선언된다. 수식어 **const**와 함께 쓰인 성원함수들과 연산자들은 고정 및 비고정객체에 의해 리용될수 있다. 이 수식(qualification)이 없는 성원함수들은 고정객체에 의해 리용될수 없다. 다음의 실례에서 고정객체 OneHalf는 insert()촉진자를 호출할수 있지만 Extract()촉진자는 호출할수 없다.

```
const Rational OneHalf(1, 2);
OneHalf.Insert(cout);//옳음
OneHalf.Extract(cin); //틀림
```

8.3.4 검토자성원함수들

목록 8-1의 Rational클라스의 정의에서 다음코드는 2개의 **protected**형검토자성원함수들을 선언한것이다.

```
int GetNumerator() const;
int GetDenominator() const;
```

이 검토자들은 분자와 분모의 내용을 호출한다. 예약어 const는 고정객체를 호출하는데 리용된다. 그것들이 protected부분에 있으므로 의뢰기프로그람은 두개의 검토자들을 리용할수 없다. 그러나 그것들은 다른 Rational성원함수들에 의해 리용될수 있다. 례를 들면 다음과 같은 코드부분은 Rational성원함수에서는 옳지만 사용자프로그람에서는 틀린다.

```
// 성원코드에서 옳음, 사용자코드에서 틀림
Rational z;
cout << z.GetNumerator() << z.GetDenominator();
```

8.3.5 변이자성원함수

목록 8-1의 Rational클라스선언에는 다음의 protected변이자선언이 있다.

```
void SetNumerator(int numer);
void SetDenominator(int denom);
```

이 선언들은 두 성원함수들은 void형이며 파라메터로서 int형을 요구한다는것을 보여 준다. 변이자는 유리수의 분자와 분모를 설정하기 위하여 리용된다. 검토자처럼 그것들은 protected부분안에서 정의되였기때문에 두 변이자는 사용자프로그람에서 리용할수 없다. 그러나 변이자는 다른 Rational성원함수들에 의해 리용될수 있다.

변이자들이 자료성원들을 수정하기때문에 그것들은 const예약어를 리용할수 없다. 그래서 변이자는 일단 객체가 구축되면 const객체로 리용될수 없다.

8.3.6 자료성원들

자료성원들은 대체로 private부분에서 선언한다. 그러므로 의뢰기프로그람이나 파생된 클라스가 자료성원을 호출하고 변경하려면 공동성원검토자와 변이자를 리용해야 한다. 정보은폐화는 자료성원들의 안전성을 담보하며 또한 의뢰자에 의해 진행된 코드를 변화시킴이 없이 서고를 갱신하거나 수정할수 있게 해준다. 서고가 갱신되면 사용자는 그 코드들을 다시 결합하기만 한다.

비const가 아닌 자료성원들은 그의 클라스선언에서 초기화될수 없다. 구축자들은 자료성원들의 초기화를 실현하는데 리용된다. 객체의 매 자료성원은 구축자에 의해 적당한 값을 초기화한다. 만일 모든 구축자들이 자료성원들을 적당한 값으로 설정한다면 다른 성원함수들과 연산자들이 적당한 값을 가지는가를 확인할 필요는 없다.

목록 8-1에서의 Rational클라스정의는 모든 Rational객체들은 2개의 int자료성원들을 가진다는것을 보여 준다.

```
int NumeratorValue;
int DenominatorValue;
```

Numeratorvalue는 유리수의 분자이며 Denominator는 유리수의 분모이다. 2개의 자료성원들은 private부분안에서 정의되므로 그것들을 비성원함수 혹은 연산자로 직접 참조하는것은 오유로 된다. r의 분자를 현시하는 프로그람 8-1의 main()에 다음과 같은 명령문이 있다면 오유통보가 나타난다.

```
cout << "Numerator of rational number: "
  << r.Numeratorvalue << endl; //오유호출
```

오유통보는 비성원함수가 틀리게 참조되였다는것을 의미한다. 이러한 과정들은 목록 8-1에서 한 Rational클라스정의의 소개로 된다.

8.3.7 다중정의된 연산자들

Rational보조산수연산자들의 기본형태들은 목록 8-1에 주어 졌다. 이 기본형들은 연산자다중정의선언의 한가지 실례이다.

```
Rational operator + (const Rational &r, const Rational &s);
Rational operator * (const Rational &r, const Rational &s);
```

연산자는 함수와 같은 목적으로 리용되도록 하기 위해 다중정의된다. 그것들은 다른 함수들과 연산자들이 형연산자를 호출할수 있게 대면부를 서술해 준다. 예약어 **operator**는 연산자가 다중정의되였다는것을 의미한다.

연산자형선언 혹은 정의가 주어 지면 처음으로 그 연산자에 대한 대면부가 서술된다. 그 대면부서술은 연산자의 귀환형부터 시작한다. 그것이 클라스정의밖에서 정의되는 성원연산자일지라도 그 연산자의 귀환형뒤에 예약어 **operator**가 붙는다(만일 성원연산자가 그의 클라스정의밖에서 정의된다면 클라스다음과 유효범위해결연산자 (::)는 예약어 **operator**를 앞에 놓는다). 예약어 **opertor**뒤에 다중정의된 실제적인 연산자가 놓인다. 그 대면부는 연산수의 선언으로 완성된다. 연산수들의 선언은 파라메터목록으로서 주어 졌다.

유리수산법의 결과가 유리수값이기때문에 Rational산수연산자정의들은 그의 귀환형으로서 Rational을 가진다. 2개의 Rational산수연산과 정의는 두개의 연산수(파라메터)들을 지적한다. 왼쪽연산수가 제일 처음 선언되며 그다음 오른쪽연산수가 선언된다. 그 연산수들은 연산의 결과로 수정될수 없으므로 연산수들은 **const**파라메터로써 정의된다. 그 연산수들은 이러한 충분한 리유로 하여 **const**참조파라메터로 정의된다(만일 그것들이 값파라메터였다면 객체의 복사가 산수연산자안에서 참조되고 넘겨 진다).

Rational객체를 위한 삽입연산자를 다중정의하는 능력도 쏘프트웨어기술자들이 stdio서고에 비하여 그것을 iostream서고를 더 좋아 하는 리유에 대한 한가지 실례이다. iostream서고안에서 설계된 기능은 모든 입력과 출력도구들에 맞게 대면부가 모순이 없도록 출력과 얻기연산자들을 다중정의하는것이 가능하게 한다. 이 표현은 stdio서고를 가지고서는 불가능하며 그것이 모든 입력과 출력에 오직 기초형들과 문자렬들만을 요구하도록 한다. 삽입과 추출연산자들의 기본형은 또한 2개의 연산수를 지적한다.

```
ostream& operator << (ostream &sout, const Rational &r);
istream& operator >> (istream &sin, Rational &r);
```

입력연산자와 추출연산자들의 왼쪽연산수들은 참조파라메터로서 선언된다. 왼쪽연산수들은 참조파라메터이다. 왜냐하면 Rational객체의 삽입과 추출이 흐름을 변화시키며 이런 변화가 의뢰자프로그람에 있어서 영구적이므로 참조파라메터로 된다. ostream은 모든 출력흐름의 기초클라스이다. istream은 입력흐름에 대한 기초클라스이다. 이 기초클라스들을 리용하여 연산자들은 표준흐름, 파일흐름 그리고 기억기관리 문자렬흐름들로 호출될수 있다.

이러한 입력과 추출연산자들은 참조귀환을 수행한다. 귀환형에 대한 지적자 &는 귀환형을 지적한다.

표준귀환에서 함수 혹은 연산자의 귀환값은 되돌리는 값을 창조하는 림시객체이다. 림시객체의 존재는 호출식에 의해 규정된다. 참조귀환에서는 림시객체가 만들어 지지 않는다. 귀환값은 호출을 포함하는 연산자나 함수의 유효범위를 포함하는 속성을 가진 객체에 제공된다. 성원연산자함수에서 객체호출은 이러한 속성을 가진다. 참조파라메터 역시 이 속성을 가진다. 참조귀환으로 하나의 Rational객체출력 혹은 얻기는 출력 혹은 얻기연산의 더 큰 부분으로 될수 있다. 례를 들면 다음과 같은 코드부분에서 처음은 Rational r가 표시되고 그다음 행바꾸기문자가 표시된다.

```
Rational r(1, 2);
cout << r << end1;
```

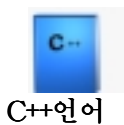
삽입명령문은 정확히 동작한다. 왜냐하면 cout<< r의 결과가 실제로 cout이며 어떤 림시흐름이 아니기때문이다. 그러므로 다음출력은 조종자 endl을 리용할것을 요구한다. 값이 되돌려 지면 end1출력명령문은 림시복사를 진행한다. 이것은 요구된 결과가 아니다.

참조귀환을 하는 기능이 없다면 출력명령문은 void형으로 되여야 하며 출력표현을 두개로 가를수 있다.

```
cout << r;
cout << endl;
```

참조값이 표준귀환보다 더 효과적이다. 림시객체가 창조될 필요는 없다. 클라스를 리용하는것은 Rational보다 더 복잡한 객체를 명백하게 하여 주기때문이다.

참조값이 참조되돌림값을 적용하기 위해 국부변수를 리용한다면 콤파일프로그람은 오유통보를 발생시킨다. 만일 그렇지 않다면 함수호출인용에서 객체의 리용은 정의되지 않은 동작으로 평가된다. 그 동작은 정의되지 않는다는것이다. 왜냐하면 귀환값을 리용하는것이 이미 해방된 활성화레코드기억기로 되기때문이다.

C++언어

구조체

클라스구조는 C의 struct구조를 보다 일반화한것이며 그것은 자료성원들을 포함할수도 있고 성원함수나 연산자들을 포함할수 있다. 다시 말하면 protected 혹은 private부분일수 없다. C언어는 정보를 은폐하는데서 C++보다 기능이 약하다.

C++도 struct구조를 가진다. 그러나 C++에서는 C의 struct구조보다 클라스구조를 더 좋아한다. C++ struct는 자료성원들, 성원함수들 그리고 연산자들이 가능하다면 접근지정자도 쓸수 있게 한다. 그 struct구조는 public인 기억호출지시에서만 클라스구조와 차이난다. struct구조는 C++프로그람에서 좀처럼 리용되지 않는데 그것은 private의 기정접근지정과 함께 클라스구조가 정보은페화의 원리를 더 훌륭히 지원하기때문이다.

문제

1. 략자 ADT의 의미를 말하시오.
2. 콤파일러에 의해 지원되는 복사구축자가 수행하는것은 어떤 복사인가?
3. 2원산수연산자의 다중정의된 보조연산자를 리용하는것이 가장 좋은것으로 되는 리유를 설명하시오.
4. 데카르트공간에서 하나의 점에 대한 ADT를 설계하시오. 될수록 유연하게 ADT를 만드시오.
5. 2개의 클라스들이 꼭 같은 이름을 가진 성원함수들을 가질수 있는가?
6. 왜 출력과 입력연산자들이 참조귀환을 수행하여야 하는가를 설명하시오.
7. 다음과 같은 클라스선언이 있다.

```
class Obj {
    public:
        Obj( );
        int F1(int x);
        void F2(int x);
        void F3(int x) const;
    protected:
        void F4(int x);
    private:
        void F5(int x);
        int Count;
};
```

다음의 프로그람이 옳은가? 만일 틀린다면 무엇이 틀리는가를 설명하시오.

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    Obj Object1;
    Obj Object2;
    Object1.Count = 0;
    Object2.Count = 0;
    return 0;
}
```

8. 7번 문제의 클라스선언과 다음의 프로그람을 고찰하시오.

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    Obj Object1;
    Object1.F1(3);
    return 0;
}
```

우의 프로그람이 정확한 C++프로그람인가? 아니라면 리유를 설명하시오.

9. 7번 문제의 클라스선언과 다음의 프로그람을 고찰하시오.

```
#include <iostream>
#include <string>
using namespace std;
```

```
int main() {
    Obj Object1;
    Object1.F4(3);
    return 0;
}
```

우의 프로그람이 정확한 C++프로그람인가? 아니라면 리유를 설명하시오.

10. 문제 7의 클라스선언과 다음의 프로그람을 고찰하시오.

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    Obj Object1;
    Object1.F3(3);
    return 0;
}
```

7번 문제의 프로그람이 정확한 C++프로그람인가? 아니라면 그 리유를 설명하시오.

11. 모듈러계수기에 관한 ADT를 설계하시오. 하나의 모듈러계수기는 증가계수하는 계수기이다. 증가계수할 때 계수기의 값이 최대값에 이르면 계수기는 0으로 다시 초기화된다. ADT에 관한 클라스를 선언하시오.
12. 계수기에 관한 ADT를 설계하시오. 계수기는 증가 혹은 감소계수하는 계수기이다. 증가계수하는 경우 계수기가 최대값에 이르면 계수기는 0으로 된다. 감소계수하는 경우 계수기가 0이면 최대값으로 된다. ADT에 관한 클라스를 선언하시오.
13. 과학적인 수표시법에 관한 ADT를 설계하시오(혹은 2.34×10^{-5}형태의 수자). ADT에 적합하다고 생각하는 클라스의 선언과 보조함수들의 형변환만을 정의하시오.

8.4 유리수클라스의 실현부

유리수서고의 서술은 성원함수의 정의도 보조연산자들의 정의도 포함하지 않는다. 목록 8-2부터 8-7까지 rational.h에 서술된 함수들과 연산자들의 가능한 수행을 보여 주었다.

그 목록들을 조사하면 Rational성원함수들의 정의에서 정의되는 함수의 이름이 항상 클라스이름과 유효범위해결연산자로 시작된다는것을 알수 있다. 례를 들면 Rational변이자 SetNumerator()의 정의는 다음과 같다.

```
void Rational::SetNumerator(int numer) {
    Numeratorvalue = numer;
}
```

유효범위의 식별이 필요하다. 왜냐하면 같은 성원이름과 대면부를 리용하는 다른 클라스들에서 원천파일이 성원함수와 연산자의 정의를 포함하고 있기때문이다. 유효범위해결이 없다면 적당한 클라스의 정의는 불가능하게 된다.

성원의 정의에서 기정값으로 참조되는 성원함수나 연산자가 좀 차이나는것이 있다. 성원함수에서 국부적이지 않은 클라스유효범위이다. 그러므로 성원의 정의에서 클라스의 참조성원에 클라스이름과 유효범위해결연산자를 련결하여 리용하는것이 필요하다.

경험

정확한 연산자

유리수더하기에서 연산자 +기호는 필요하지 않지만 사용할수도 있다. 더하기에 다른 연산자를 리용하면 사용자프로그람작성자에게 혼란을 줄수 있다. 이것은 다른 연산자들의 이름을 혼돈하는것과 같은 결과를 초래할수 있다.

류사한 연산자를 리용하여 연산의 우선순서를 유지할수 있다. 연산자다중정의에서 새로운 문법은 관습적인 우선권과 같다. 실례로 Rational객체 s, t 그리고 u를 리용하는 다음과 같은 2개의 표현들은

```
s + t * u
s + (t * u)
```

과 같다. *가 +보다 더 높은 우선권을 가지므로 괄호는 항목에서 필요없다.

8.4.1 구축자정의

목록 8-2에서 두개의 구축자들은 실제적인 작업을 하기 위하여 변이자 SetNumerator()와 SetDenominator()를 리용한다.

목록 8-2. rational.cpp로부터 Rational구축자의 실현부

```
#include <iostream.h>
#include <string.h>
#include "rational.h"
using namespace std;
Rational::Rational() {
    SetNumerator(0);
    SetDenominator(1);
}
Rational :: Rational(int numer, int denom) {
    SetNumerator(numer);
    SetDenominator(denom);
}
```

처음 기정구축자가 정의되며 그것은 유리수 0/1이 표시되도록 새로운 객체를 초기화하며 계속하여 파라메터 0과 1로서 변이자를 호출한다.

```
SetNumerator(0);
SetDenominator(1);
```

변이자 SetNumerator()와 SetDenominator()의 리용은 기본자료성원의 실현부를 서로 분리시킨다. 정보은페화원리는 여러개의 함수들과 연산자들이 국부적인 자료성원들을 처리할수 있게 한다. 이러한 국부화는 일반적으로 자료를 표현하는데서 더 간단한 방법이다.

목록 8-2에 정의된 2번째 구축자는 변이자 SetNumerator()와 SetDenominator()를 호출할 때 numer와 denom의 값을 리용한다.

```
SetNumerator(numer);
SetDenominator(denom);
```

이 구축자에서 두번째 파라메터는 선택적인 파라메터이다. C++언어는 기정값을 한번만 선언할수 있게 한다. 그러므로 기정값의 선언은 클라스정의나 성원함수의 정의에서 해야 한다. 클라스정의에서 기정값을 지적해야 한다.이러한 과정은 의뢰기프로그람작성자에 의해 검사된 ADT서고의 부분에 있다.

8.4.2 검토자정의

앞에서 고찰한 바와 같이 Raitonal클라스는 정보은폐화를 지원하며 자료성원에로의 참조를 조종하는 검토자와 변이자함수도 제공한다.

목록 8-3. rational.cpp의 Rational검토자와 변이자

```
//분자를 얻는다.
int Rational::GetNumerator() const {
    return NumeratorValue;
}
//분모를 얻는다.
int Rational::GetDenominator() const {
    return Denominatorvalue;
}
//분자를 설정한다.
void Rational::SetNumerator(int numer) {
    NumeratorValue = numer;
}
//분모를 설정한다.
void Rational::SetDenominator(int denom) {
    if(denom != 0) {
        DenominatorValue = denom;
    }
    else {
        cerr << "Illegal denominator: "<< denom << "using 1" <<end1;
        DenominatorValue = 1;
    }
}
```

목록 8-3에서 검토자 GetNumerator()와 GetDenominator()의 정의는 간단하다. 유리수의 분자는 자료성원 Numeratorvalue에 보관된다. 그러므로 GetNumerator()는 자료성원의 현재값을 간단히 되돌

린다.

```
return NumeratorValue;
```

분모가 자료성원 denominatorvalue에 보관되므로 GetDenominator()는 자료성원의 현재값을 되돌린다.

```
return DenominatorValue;
```

Numeratorvalue와 Denominatorvalue를 참조하는 함수의 본체에서는 접근연산자나 유효범위해결연산자를 리용할 필요가 없다. 참조는 검토자가 호출한 객체들의 자료성원으로 된다.

8.4.3 변이자정의

목록 8-3에서의 변이자 SetNumerator()의 SetDenominator()정의는 간단하다. 이 매 함수들은 Numeratorvalue와 Denominatorvalue의 새 값으로 리용되는 하나의 int파라메터를 가지고 있다. 실례로 SetNumerator()는 Numeratorvalue를 설정하기 위해 Numer파라메터를 리용한다.

```
NumeratorValue = numer;
```

옹근수값은 유리수의 분자로 되여 SetNumerator()는 정확한 동작을 할수 없다. 그러나 변이자는 이러한 문제를 해결한다. 구체적으로 0이라는 분모값은 적당치 않으므로 SetDenominator()는 이 값을 리용하기전에 분자값을 검사한다.

```
if (denom != 0) {
    DenominatorValue = denom;
}
else {
    cerr << "Illegal denominator: " << denom
      << "using 1" << end1;
    DenominatorValue = 1;
}
```

만일 요구된 값이 0이 아니라면 그 값은 자료성원 Denominatorvalue를 설정하는데 리용된다. 만일 요구된 값이 0이라면 함수는 오유통보를 표시하며 분모값으로 1을 리용한다. SetDenominator()에 의해 발생한 오유처리는 례외처리이다. C++는 례외처리를 위한 시도-내보내기-포착(try-throw-catch)이라는 기구를 제공한다. 이 기구부는 부록 4에서 언급하였다.

8.4.4 산수촉진자의 정의

목록 4-4에서 두개의 산수촉진자는 유리수산법의 수행을 보여 준다. 새 촉진자들의 파라메터는 왼쪽연산수가 객체를 호출하게 하는 오른쪽연산수이다.

목록 8-4. Rational.cpp의 산수촉진자

```
Rational Rational :: Add(const Rational &r) const {
    int a = GetNumerator();
    int b = GetDenominator();
```

```
    int c = r.GetNumerator();
    int d = r.Denominator();
    return Rational(a*d + d*c, b*d);
}
Rational Rational :: Multiply(const Rational &r) const {
    int a = GetNumerator();
    int b = GetDenominator();
    int c = r.GetNumerator();
    int d = r.Denominator();
    return Rational(a*c, b*d);
}
```

례를 들면 다음과 같은 부분은 파라메터 y를 가지는 x의 성원함수 Add()를 호출하여 Rational객체 x와 y를 더한다. 합해 진 결과는 Rational객체 Z를 구축하는데 리용된다.

```
Rational x(1, 2);
Rational y(1, 3);
Rational z = x.Add(y) ; // 1/2 + 1/3 은 5/6
cout << z << end1;
```

성원함수 Add()의 정의는 2개의 int형객체 a와 b를 정의하는것으로부터 시작된다.

```
int a = GetNumerator();
int b = GetDenominator();
```

이 객체들은 호출되는 객체들의 분자와 분모의 복사를 진행한다. 우의 실례에서 a는 1로, b는 2로 초기화된다. 그다음 함수는 2개의 int객체 c와 d를 정의한다. 이 객체들은 파라메터의 분자와 분모의 복사를 진행한다. 앞의 실례에서 c는 1로, d는 3으로 초기화된다.

```
int c = r.GetNumerator();
int d = r.GetDenominator();
```

이 4개의 객체들은 유리수 합 a/b + c/d를 계산하는 Rational객체를 정의하는데 리용된다. 구축된 객체는 촉진자의 귀환값으로 리용된다.

```
return Rational(a*d + b*c, d*b);
```

합을 구하는 Rational객체의 구조에는 이름이 없다. 귀환형은 파라메터 a*d + b*c 를 가진 구축자를 직접 호출한다. 그다음 명령은 이름을 리용하여 이 값을 참조할 필요가 없으므로 이 호출을 리용할수 있다.

촉진자 Multiply()의 정의는 Add()과 비슷하다.

8.4.5 입력과 출력정의

촉진자 Insert()와 Extract()의 정의는 목록 8-5에 주었다.

목록 8-5. rational.cpp의 흐름촉진자

```
void Rational::Insert(ostream &sout) const {
    sout << GetNumerator() << '/' <<GetDenominator();
    return;
}
void Rational::Extract(istream &sin) {
    int numer;
    int denom;
    char slash ;
    sin >> numer >> slash >>denom;
    SetNumerator (numer);
    SetDenominator(denom);
    return;
}
```

촉진자 Insert()는 ostream참조파라메터 sout출력흐름을 가진다. Rational객체를 출력하는 처리는 아주 간단하다. 분자를 표시하고 /와 분모를 표시한다.

```
sout << GetNumerator () << '/' << Getdenominator();
```

정보은페화원리가 유리수의 표시에 영향을 준다는데 주의하시오. 호출객체들의 분자와 분모는 검토자 GetNumerator()와 GetDnominator() 리용하여 호출될수 있다.

목록 8-5에서 Extract()촉진자함수의 정의는 간단한 istream참조파라메터 sin을 지정한다. 파라메터 sin은 호출되는 객체에서 입력되는 유리수의 원천흐름이다. 그 정의는 입력되는 유리수의 분자에 대응하는 int값을 입력하는것으로부터 시작된다.

```
sin >> numer >> slash >> denom;
```

한 문자가 입력되였다면 이 경우에 분모와 분자를 분리시키는 빗선(/)을 넣어야 한다. 례외처리는 입력된 문자가 빗선(/)이라는것을 확인한다. 마지막으로 요구되는 분모의 값이 int객체 denom에 보관된다.

촉진자 Extract()는 다음으로 호출되는 객체의 자료성원들을 재설정하기 위하여 SetNumerator()와 SetDenominator()의 호출시 2개의 입력값을 리용한다.

```
SetNumerator (numer);
SetDenominator (denom);
```

분모의 새로운 값을 검사하는데 Extract()가 필요없다. SetDenominator()로 그 정확성을 검사할 필요가 없다.

8.4.6 보조적인 산수연산자정의

목록 8-6에서 다중정의된 2개의 산수연산자들은 필요한 동작을 수행하기 위하여 적당한 촉진자를 리용한다.

목록 8-6. ratinal.cpp의 산수연산자

```
Rational operator + (const Rational ) &r.
  const Rational &s) {
    return r.Add(s);
}
Rational operator *(const Rational &r,
  const Rational &s) {
     return r.Multiply(s);
}
```

실례로 연산자 +는 촉진자에 대한 파라메터로서 오른쪽연산수 s를 리용하는 왼쪽연산수 r의 더하기촉진자 Add()를 호출한다. r.Add(s)는 r와 s의 합을 계산한다. 그다음 해당한 귀환값을 돌려 준다. 이렇게 연산자 +는 1개의 명령으로 완성할수 있다.

```
return r.Add(s);
```

이와 같이 연산자 *는 단순한 명령으로 연산수 r와 s의 곱하기연산결과를 돌려 주는 동작을 수행한다.

```
return r.Multiply(s);
```

앞에서 고찰한바와 같이 산수연산자들은 필요없다. 연산자로써 간접적으로 촉진자들을 호출하기보다 직접 연산자의 동작을 수행하는 촉진자들을 리용할수 있다. 그러나 촉진자를 직접 리용하는것은 쉽게 재리용할수 있는 프로그람을 작성할수 있게 하는 서고에서는 애매한 점이 있다. 의뢰자의 과제를 연산자로 더 쉽게 완성할수 있다. 의뢰자는 연산자를 리용하여 유리수의 값을 계산할수 있다. 그러므로 산수연산자들은 서고의 중요한 부분으로 된다. 서고에서 흐름연산자의 실행을 고찰해 보아야 한다.

경험

다중정의하기전에 생각하자.

연산자를 다중정의하려면 그것이 어떻게 리용될것인가를 연구해야 한다. 기초객체의 기본설정에서 연산자를 주시해 보고 다중정의하는데서 비슷한 동작이 있는가 확인한다.

8.4.7 보조적인 흐름연산자정의

목록 8-7에서 출력연산자정의는 두 연산수를 요구한다. 왼쪽연산수 sout는 출력하기 위한 목적출력흐름이다. 오른쪽연산수 r는 현시하려는 Rational객체이다.

목록 8-7. rational.cpp의 흐름연산자

```
ostream& operator << (ostream &sout, const Rational & r) {
    r.Insert(sout);
    return sout;
}
istream& operator >>(istream &sin, Rational &r) {
    r.Extract(sin);
```

```
        return sin;
    }
```

산수연산자의 다중정의와 마찬가지로 출력연산자의 다중정의도 간단하다. 출력은 촉진자의 파라메터로서 출력흐름 sout를 가지는 r의 public함수 Insert()를 리용하여 진행된다.

```
r.Insert(sout);
```

Ostream sout에로의 참조귀환은 연산자의 동작을 수행한다.

```
return sout;
```

8.3.7에서 언급한것처럼 참조귀환은 Rational입력이 간단한 표현으로 다른 값을 련속 입력하게 한다.

목록 8-7의 출력다중정의에서 operator >>( )의 정의도 간단하다. 파라메터로서 왼쪽연산자입력흐름 sin을 가지는 촉진자 Extract( )를 리용하여 오른쪽연산수 r를 재설정한 다음 istream sin에로의 참조가 돌려 진다. 참조귀환은 Rational출력이 단순한 표현으로 련속 진행하게 한다.

8.5 복사구축과 성원값주기, 해체

C++가 자동적으로 Rational클라스의 복사구축자와 성원값주기연산자를 리용가능하게 한다는데 대해서는 고찰하였다.

```
Rational r(1, 2);
Rational s( r);          //s는 r에서 구축된 복사이다.
Rational t;
t = r;                   //t 는 r 로 값주기된 성원이다.
```

r의 복사는 s를 창조하고 t를 수정하는데 이러한 성원들을 리용한다. 콤파일러는 목적객체에서 원천객체에 성원별복사를 진행한다. 실례에서 r는 원천객체이고 s는 구축하려는 목적객체이며 t는 값주기의 목적객체이다. 성원별복사에서 원천객체의 자료성원들은 목적객체들이 자료성원들과 대응하게 비트별로 복사된다. 성원별복사를 얕은 복사(shallow copying)라고 한다.

C++는 자동적으로 클라스형객체에 관한 해체자(destructor)성원함수도 만든다. 객체의 해체자는 객체를 해체할 때 자동적으로 호출된다. 해체자는 해당객체에 대한 필요한 지우기처리를 수행한다. 해체자에 이미 제공되여 있는 콤파일러는 그밖의 기능을 수행하지 않는다.

이렇게 해체는 보충적으로 제공되는 기능이며 그것은 창조되여 있는 객체에 대하여 필요한 처리를 수행한다. 해체자는 기호(~)와 클라스이름을 리용하여 선언할수 있다(다른 문법에서는 ~가 비트연산자이다). Rational클라스의 해체자는 ~Rational()이다.

경험

3개의 조

클라스는 객체들이 동적인 기억기할당을 직접 리용하는 자료성원들을 가지는 경우에만 복사구축자, 값주기연산자, 그리고 해체자의 정의를 요구한다. 그밖의 경우에는 자동적인 판본이 일반적으로 적합하다. 클라스의 설계에서 이 세 성원들중 어느 하나를 정의할수도 있으며 그것들을 모두다 정의할수 있다.

C++는 해체자선언에서 두가지 제한조건이 있다. 해체자는 파라메터를 가질수 없으며 귀환값이 없다는것이다.

또한 복사구축자에서도 다음과 같은 조건이 있다. 복사구축자는 하나의 형식파라메터를 가지며 이 파라메터는 클라스에서 같은 형태를 가진다. 형식파라메터는 const참조로 되여야 한다. 값주기연산자에 대한 파라메터도 이와 같다. 복사구축자나 해체자와 달리 값주기연산자는 귀환형을 가진다. 귀환형은 값주기한 클라스의 참조로 된다.

Rational복사구축자, 값주기연산자, 해체자를 명백하게 정의하였다면 이 3개의 성원의 미리선언을 포함하기 위해 rational.h머리부파일의 대면부를 변경시켜야 한다.

```
class Rational {
    public:
        Rational(const Rational &r);
        Rational& operator (const Rational &r);
        ~Rational();
}
```

우의 클라스정의에서 성원값주기연산자에 대하여 문법적으로 보면 이상한 점이 있다. 값주기연산자는 2개의 연산수를 가지며 이 선언은 오직 하나의 연산수만을 지적한다. 이 문법은 왼쪽연산수가 값주기연산자를 호출하는 객체라는것을 알고 있기때문에 리용되였다. 그러므로 오른쪽연산수의 선언만이 필요하다. 하나의 값을 돌려 주는 연산자는 다음과 같은 코드부분에서 더 긴 식으로 표시될수 있다.

```
Rational x(1, 2);
Rational y;
Rational z;
z = y = x;
```

값주기연산자가 참조귀환을 진행하므로 연산은 더 효과적으로 수행된다.

목록 8-8은 Rational복사구축자, 값주기연산자, 해체자를 명백하게 정의한다. 이 정의들은 콤파일러에 의하여 같은 기능을 제공한다. 복사구축, 값주기, 해체의 기초개념을 정확히 리해해야 한다. 11장에서 이 개념들을 다시 학습하게 되며 거기에서 명백하게 주어 진 이 세 성원을 요구하는 ADT를 진행하게 된다.

목록 8-8. Rational복사구축자, 값주기연산자, 해체자의 정의

```
Rational Rational::Add(const Rational & r)  {
    int a = r.GetNumerator( );
    int b = r.GetDenominator( );
    SetNumerator(a);
    SetDenominator(b);
}
Rational::~Rational( ) {
}
Rational& Rational :: operator=(const Rational & r) const {
```

```
        int a = r.GetNumerator( );
        int b = r.GetDenominator( );
        int c = r.GetNumerator( );
        SetNumerator(a);
        SetDenominator(b);
        return *this;
    }
```

Rational복사구축자의 정의는 간단하다. 구축되는 객체는 구축자에 파라메터 r를 복사한다. 그렇게 하기 위하여 처음 r의 분자와 분모는 r의 성원함수 GetNumerator( )와 GetDenominator()를 호출하여 국부변수 a와 b를 초기화하는데 리용된다.

```
int a = r.GetNumerator( )
int b = r.GetDenominator( );
```

변수 a와 b는 그다음 성원함수 SetNumerator( )와 SetDenominator( )의 파라메터로 구축되여 있는 객체로 리용된다.

```
SetNumerator(a);
SetDenominator(b);
```

목록 8-8에서 Rational해체자의 정의는 더 간단하다. 요구되는 기능이 없으므로 해체자의 본체에는 아무것도 없다.

목록 8-8에서 Rational값주기연산자를 해석할 때 연산자가 리용된다면 목적객체가 목적자료성원을 동작하도록 연산자를 호출한다는것을 알고 있어야 한다. 값주기연산자는 처음에는 복사구축자와 비슷한 기능을 수행한다. 파라메터 r의 분자와 분모의 값은 목적객체의 분자와 분모를 설정하는데 리용한다.

```
int a = r.GetNumerator( )
int b = r.GetDenominator( );
SetNumerator(a);
SetDenominator(b);
```

Rational값주기연산자에 관한 귀환값은 표현 *this이다. C++는 호출되는 성원함수의 객체주소인 예약어 this를 관리한다. 이 문법에서 *는 참조해제(dereferencing)형연산자이다. 예약어 this의 참조해제형연산자는 단항연산자이다. 참조해제연산자는 주소에 적용될 때 그 주소에 객체의 값을 만든다. 표현 *this로 호출된 성원함수의 객체를 표현하며 그것은 값주기연산자에 의해 귀환되는 정확한 값이다. 복사구축자, 값주기연산자 그리고 프로그람안에서 해체자의 동작을 설명하기 위해 작고 간단한 클라스 c를 정의한다. 목록 5-9에서 클라스의 정의를 기정구축자, 복사구축자, 해체자, 값주기연산자 string형자료성원 name을 지적한다.

목록 8-9. 기정구축자, 복사구축자, 값주기연산자, 해체자를 가진 클라스 C

```
class C{
    public:
```

```
        C() {
            name = 5;
            cout << name << ": default constructed" <<end1;
        }
        C(const C &c) {
            name = 5;
            cout << name << ": copy constructed using " << c.name <<end1;
        }
        ~C( ) {
            cout << name << ": destructed" <<end1;
        }
        C& operator = (const C &c) {
            cout <<name << ": assigned using " << c.name <<end1;
            return *this;
        }
    private:
        string name;
}
```

성원함수기본형의 정의뿐아니라 클라스정의는 또한 함수의 정의를 포함한다. 이러한 결합은 간단히 언급된다. 실현부와 대면부를 갈라 보아야 한다.

클라스 C의 구축자는 두가지 기능을 수행한다. 그것들은 전체 객체 S의 현재값으로 자료성원 name을 확정하고 그것들이 왜 호출되였는가를 지적하는 통보문을 현시하기 위하여 그 값을 리용한다. 류사하게 해체자와 값주기연산자는 자기들이 호출당한 리유를 지적하는 통보문을 표시한다. 클라스 C는 다음과 같은 프로그람부분에서 리용된다.

```
string s = "w";
C w;
int main() {
    s = "x";
    C x;
    {
        s = "y";
        C y;
    }
    s = "z";
    C z(w);
    x = w = z;
    return 0;
}
```

프로그람의 출력을 조사함으로써 클라스 C성원함수가 어떻게 언제 기동하는가를 알수 있다. 이 출력은 다음과 같다.

```
w: default constructed
x: default constructed
y: default constructed
y: destructed
z: copy constructed using w
w: assigned using z
x: assigned using w
x: destructed
w: destructed
```

이 출력은 이 부분에 의하여 정의된 첫 C객체가 대역객체로 변한다는것을 보여 준다. 다음 정의된 두개의 C객체들은 x와 y이다. 이 객체들은 함수 main()에 대하여 국부적이다. 객체 y는 main()의 내부블로크에서 정의된다. y의 유효범위는 그 내부블로크안으로 제한된다. 출력은 y의 해체자가 블로크가 끝나면 자동적으로 호출되는것을 지적한다. 함수 main()은 객체 w로부터 객체 z에로 복사구축을 계속한다. 해체가 끝나기전에 함수 main()은 두개의 값주기를 진행한다.

```
x = w =z;
```

예측할수 있는바와 같이 출력은 값주기연산자가 오른쪽으로부터 왼쪽에로 값주기된다는것을 지적한다. 객체 x와 z가 main()에 대하여 국부적이므로 그것들은 함수가 끝나면 해체된다. 특히 z는 x후에 정의되였으므로 x보다 먼저 해체된다.

Global객체는 프로그람이 끝나기전에 즉시 해체되며 조종건은 조작체계에로 돌아 간다. 마지막프로그람출력은 w가 해체되였다는것을 지적한다.

련습에서 c객체의 출력과 더하기연산자를 다중정의하여 복사구축자가 식에서 어떤 역할을 하는가를 알수 있다. 다음부분에서 모조란수렬서고를 진행한다.이 서고는 간단한 추측유희의 진행에 리용된다.

문 제

14. 다음과 같은 프로그람이 있다.

```
#includes <iostream>
#include <string>
using namespace std;
int count = 0;
    class obj {
        public:
        obj();
        ~obj();
    };
obj() {
    ++count; cout <<count <<end1;
```

```
    }
    ~obj() {
        --count; cout << count << end1;
    }
    int main() {
        obj A;
        {
            cout << "begin block " <<end1;
            obj B;
            cout << "end block" <<end1;
        }
        return 0;
    }
```

이 프로그램의 결과는 무엇인가?

15. 다음과 같은 프로그램이 있다.

```
    #include <iostream>
    #include <string>
    using namespace std;
    class thizbin {
        public:
            thizbin();
            thizbin(int a);
            thizbin(int a, int b, int c);
            print(const string &msg);
        private:
            int i, j, k;
    };
    thizbin() {
        i = j = k = 0;
    }
    thizbin(int a) {
        i = j = k = a;
    }
    thizbin(int a, int b, int c) {
        i = a; j = b; k = c;
    };
    print (const string &msg) {
        cout << msg << ":" <<end1;
```

```
        cout << "i =" << i << end1;
        cout << "j=" << j << end1;
        cout << "k =" << k << end1;
    }
    int main() {
        thizbin A;
        thizbin B(47);
        thizbin C(9, 11. 47);
        A.print("A object");
        B.print("B object");
        C.print ("C object");
        return 0;
    }
```

이 프로그람의 결과는 무엇인가?

16. 다음과 같은 프로그람이 있다.

```
    class CounterObj {
        public:
            int F1(int x);
            void F2(int x);
            void F3(int x) const;
        private:
            void F4(int x);
            int Count;
    };
```

자료성원 Count를 수정할수 있는 성원함수는 어느것인가?

17. 다음과 같은 클라스선언과 함수기본형이 있다.

```
    class Obj {
        public:
            Obj();
            int F1(int x);
            void F2(int x);
            void F3(int x) const;
        protected:
            void F4(int x);
        private:
            void F5(int x);
            int Count;
    };
```

int F6(int y);

성원함수 F1()은 성원함수 F2()를 호출할수 있는가. 왜 그런가?
성원함수 F2()는 성원함수 F4()를 호출할수 있는가. 왜 그런가?
성원함수 F4()는 성원함수 F2()를 호출할수 있는가. 왜 그런가?
성원함수 F6()은 성원함수 F4()를 호출할수 있는가. 왜 그런가?
성원함수 F4()는 성원함수 F3()을 호출할수 있는가. 왜 그런가?
성원함수 F3()은 성원함수 F2()를 호출할수 있는가. 왜 그런가?

18. 모드계수기(modulo counter)에 관한 ADT를 설계하고 완성하시오. 모드계수기는 올리계수할수 있는 계수기이다. 올리계수할 때 계수기가 최대값에 이르면 계수기의 값은 0으로 된다.
19. 쌍방향계수기에 관한 ADT를 설계하고 완성하시오. 쌍방향계수기는 내림 혹은 올리계수를 할수 있는 계수기이다. 올리계수할 때 계수기가 최대값에 이르면 계수기는 0으로 된다. 내리계수할 때 계수기가 0에 이르면 최대값으로 된다.
20. 과학적표시법에 의한 수에 관한 ADT를 설계하고 완성하시오(실례로 $2.34*10^{-5}$). ADT에 대하여 적합하다고 생각되는 임의의 보조적인 함수들을 선언하고 실현하시오.

8.6 옹근수모조란수의 ADT

5장에서 함수 Uniform()과 Initialize seed()는 유일한 규칙적인 문자렬을 생성하기 위하여 개발되였다. 유일한 규칙적인 값이 우연적인 방식으로 발생한다. 이 절에서 규칙적인 동작을 수행하는 ADT를 객체지향적으로 더 세밀히 고찰하고 개발하려고 한다. 8.7에서 간단한 유희를 개발하는데 이 ADT를 리용한다. 이 ADT에 련관된 서고는 randint라고 한다. randint에 의해 정의된 클라스는 RandomInt이다.

함수 Uniform()과 InitializeSeed()처럼 클라스 RandomInt는 stdlib서고함수인 rand()와 srand(), 시간표준서고함수인 time()을 리용해야 한다. 그러므로 이 서고함수들의 동작을 고찰해야 한다. 매 시간에 함수 rand()가 호출되면 그것은 0~RAND_MAX사이에 포함되는 유일한 규칙적인수를 돌리는데 거기서 RAND_MAX는 stdlib서고머리부파일에서 정의된다. 함수 srand()는 파라메터로서 unsigned int를 요구하는데 그것은 기초수를 설계하기 위해 리용된다. 여러가지 기초수값을 rand()가 여러가지 규칙적인 수를 생성하도록 한다. 그리고 기초수로서 현재시간을 리용함으로써 기초수가 프로그람의 매 실행마다 달라 지게 된다. 현재시간은 시간표준서고로부터 함수 time()을 리용하게 결정할수 있다. 0인 파라메터를 가진 함수 time()은 자기의 귀환값을 시동하여 현재시간을 제공한다.

유일한 규칙적인 수행을 표현하는 클라스는 여러가지 종류의 유일한 규칙적인 동작을 수행한다(즉 1~6사이에서 혹은 1~32사이에서의). 란수렬을 표현하는 객체는 다음과 같은 방법들을 제공한다.

- 각이한 모조란수렬을 생성한다.
- 같은 모조란수렬을 재생성한다.
- 지적된 간격으로 모조란수렬을 제한한다.
- 렬에서 다음수를 생성한다.

이러한 방법들을 지원하기 위하여 여러가지 다른 방법들이 리용된다.

- 수를 구하는 관계를 설정한다.
- 란수렬에 관한 시초수를 설정한다.

- 수를 구하는 간격에서 보조값을 검사한다.
- 수를 구하는 간격에서 높은 값을 검사한다.

추가적으로 매 프로그람이 실행될 때 서로 다른 란수들을 자동적으로 발생하는 보조함수가 있어야 한다. EzRandomize()는 EzWindows서고에서 제공되는 함수이다. 표현된 자료만이 그려 지는 란수에서 끌점값으로 된다. 이 값들을 표현하기 위한 자료성원들은 정보은폐화를 제공하는 **private**부분에 배치된다.

우의 분석에 기초한 서고대면부는 목록 8-10에 주었다. 대면부를 제공하는 클라스는 RandomInt이다.

목록 8-10에서 지적한것처럼 클라스 RandomInt는 두개의 구축자를 가진다. 첫 구축자는 두개의 선택가능한 파라메터들인 a와 b를 가지는데 그것은 기정구축자로 된다. 파라메터 a와 b는 수렬들에서 구해지는 수값들까지 포함하여 그사이의 값을 지적한다.

목록 8-10.　　　　　randint서고에서 randint.h 머리부파일

```
#ifndef RANDOM_H
#define RANDOM_H
#include <stdlib.h>
class RandomInt {
    public:
        RandomInt (int a = 0, int b = RAND_MAX);
        RandomInt (int a, int b, unsigned int seed);
        int Draw();
        void SetInterval(int a, int b);
        void SetSeed(unsigned int s);
        int GetLow();
        int GetHigh();
    private:
        int Low;
        int High;
};
unsigned int EzRandomize();
#endif
```

파라메터들에 대한 기정값들은 구축되는 RandomInt객체가 Srand()를 리용하지 않고 rand()를 반복호출하여 란수값렬을 생성한다. 기정값은 0~RAND_MAX이다.

다른 RandomInt구축자는 3개의 파라메터를 가진다. 그중에서 첫 2개 파라메터는 a와 b이며 그것들은 구해 지는 수들까지 포함하여 간격을 표시한다. 세번째 파라메터는 처음에 규칙적인 수들을 생성하기 위한 기초수값이다.

다음의 코드에서 우리는 RandomInt객체들인 R, S, T를 정의한다. 객체 R는 기정값파라메터를 리용한다. 객체 S 와 T는 둘 다 란수렬에서의 수값들의 간격을 지적한다. 객체 S는 1~6까지의 수값, 객체

T는 1~32까지의 수값을 포함한다. 객체 T에 대하여 초기 기초수값은 88로 되여 있다. 객체 R와 S 는 기정구축자에 의해 구축되며 그사이에 객체 T가 다른 구축자에 의해 구축된다.

```
//0 부터 RAND_MAX 까지의 수렬
RandomInt R;
//1부터 6까지의 수렬
RandomInt S(1, 6);
//시초수 88을 리용하는 1부터 32까지의 수렬
RandomInt T(1, 32, 88);
```

변이자성원함수 Draw()는 자기의 값으로 수렬에 다음수값을 돌려 준다. 실례로 다음과 같은 코드부분은 RandomInt객체 U를 정의하고 그 렬에서 첫 5개의 성원들을 표시한다.

```
RandomInt U;
for(int i = 1; i <= 5; ++i) {
    cout << U.Draw() << end1;
}
```

그의 실현부는 rand()함수의 반복호출동작을 모방하였기때문에 출력은 5장에서 본 프로그람 5-4의 출력과 같다.

```
346
130
10982
1090
11656
```

보조적인 EzWindow서고함수 EzRandomize()의 목적은 변이자가 서로 다르게 인용되지 않고 지정되지 않은 값으로 란수렬을 설정하는데 있다. 례를 들어 다음의 코드부분이 U를 리용하는 부분대신 실행되였다고 하자. 이 부분은 EzRandomize()를 호출하고 RandomInt객체 V를 정의하여 V의 렬에서 첫 5개의 수들을 표시한다.

```
RandomInt V;
EzRandomize();
for(int i = 1; i<= 5; ++i) {
    cout << V.Draw() << end1;
}
```

호출되는 EzRandomize()은 여러개의 기초수가 V의 수렬에 작용하도록 한다. 그 결과에 V로부터 구해 지는 렬들은 U로부터 구해 지는 렬들과는 다르다.

```
15438
1866
2330
30933
```

7595

비록 우의 코드부분에서 그것을 리용하지 못하였지만 EzRandomize()는 귀환값을 생성한다. 그 귀환값은 Rand에 대한 기초수를 설정하기 위하여 Srand()에 넘겨 지는 값이다.

클라스 RandomInt는 2개의 변이자를 가진다. 변이자 SetInterval()은 모조란수를 구하기 위하여 요구되는 수값의 끝지적자를 설정하는 **int**파라메터를 리용한다. 변이자 SetSeed()는 다음수를 생성하는 기초수값을 설정하기 위한 1개의 파라메터를 리용한다.

검토자 GetLow()와 GetHeight()는 같은 규칙을 가지고 있다. 그것들은 모조란수들을 구하는 값의 최소, 최대끝지적자의 값을 돌려 준다.

8.6.1 RandomInt의 완성

RandomInt서고성원함수들의 실행파일은 RandomInt.cpp이다. 이 실행파일은 목록 8-10에 있다. 목록은 Stdlib성원함수 rand()와 srand()가 RandomInt클라스를 만든다는것을 보여 주고 있다.

목록 8-11.　　　　randint.cpp파일의 실행

```
#include <iostream>
#include <string>
#include <stdlib.h>
#include <time.h>
#include "randint.h"
using namespace std;
//RandomInt():기정란수렬
RandomInt::RandomInt(int a, int b) {
    SetInterval(a, b);
}
//RandomInt():기정란수렬
RandomInt::RandomInt(int a, int b, unsigned int s) {
    SetInterval(a, b);
    SetSeed(s);
}
//SetInterval():아래값과 웃값을 설정
void RandomInt::SetInterval(int a, int b) {
    if ( a > b) {
        cerr << "Bad random number interval: " << a << " … " << b << end1;
        exit(1);
    }
    else {
        Low = a;
        Hight = b;
```

```
    }
}
//SetSeed():렬을 설정
void RandomInt::SetSeed(unsigned int s) {
    srand(s);
}
//Draw():렬에서 다음값을 되돌리기
int RandomInt::Draw() {
    int IntervalSize = GetHigh() - GetLow() +1;
    int RandomOffset = rand() % IntervalSize;
    int Number = GetLow() + RandomOffset;
    return Number;
}
//GetLow(): 아래값을 되돌린다.
int RandomInt::GetLow() {
    return Low;
}
//GetHigh():웃값을 되돌린다.
int RandomInt::GetHigh() {
    return High;
}
```

RandomInt의 기정구축자는 정보은페화를 간단히 실현한다.

```
RandomInt::RandomInt(int a, int b) {
    SetInerval(a, b);
}
```

그 구축자는 얻어 지는 수로부터 수렬값의 끝지적자를 설정하는데 변이자 SetInterval()을 리용한다. 구축자정의에서는 파라메터들의 기정값을 가질수 있다는것을 지적하지 못한다는데 주의를 돌리시오. 그대신 클라스정의는 기정값을 정의한다. 다른 RandomInt구축자도 정보은페화를 실현하는 과정이 간단하다.

```
RandomInt::RandomInt(int a, int b, unsigned int s) {
    SetInterval (a, b);
    SetSeed(s);
}
```

이 구축자도 주어 지는 수로부터 수렬의 끝지적자값을 지적하기 위하여 변이자 SetInterval()를 리용한다. 구축자는 그다음 모조란수들에 관한 기초수를 지적하기 위하여 변이자 SetSeed()를 리용한다.

자기의 과제를 실행하기 위하여 변이자 SetInterval()은 처음 잘 알고 있는 요구되는 끝지적자값을 확인하기 위하여 파라메터들을 검사한다.

```
void RandomInt::SetInterval(int a, int b) {
```

```
        if (a > b) {
            cerr << "Bad random under interval: " << a << " ... " << b << endl;
            exit(1);
        }
        else {
            Low = a;
            High = b;
        }
    }
```

만일 끝값이 정확하다면 자료성원 Low와 High는 파라메터 값들을 리용하여 설정한다.

변이자성원함수 SetSeed()는 함수 Srand()의 호출에 자기의 파라메터를 리용한다.

```
    void RandomInt::SetSeed(unsigned int s) {
        srand(s);
    }
```

성원함수 Draw()가 모조란수의 생성에서 rand()을 리용하기때문에 SetSeed()의 호출은 생성되는 수렬에 영향을 주게 된다. Draw()함수는 요구되는 수값으로 규칙적인 모조란수를 생성하는데 여러가지 작은 관계들을 수행하게 된다. 그 단계들은 5장의 함수 uniform()와 비슷하다.

```
    int RandomInt::Draw() {
        int IntervalSize = GetHigh() - GetLow() +1;
        int RandomOffSet = rand() % IntervalSize;
        int Number = GetLow() + RandomOffSet;
        return Number;
    }
```

함수 Draw()는 처음에 객체 IntervalSize에서 간격값의 크기를 계산한다. 이를 위하여 검토자 GetLow()와 GetHigh()를 리용한다. 다음 rand()에 의해 만들어 진 란수는 IntervalSize로 되며 RandomOffSet에 값주기한다.

```
    int RandomOffset = rand() % IntervalSize;
```

즉 RandomOffset는 수값 0 ~intervalSize-1까지의 값으로 된다.

intervalSize가 2라고 하면 식 Rand() % intervalSize의 값은 0 혹은 1로 된다. 만일 RANDMAX의 값이 홀수라면 값 0과 1은 rand()가 시간의 절반만큼은 홀수, 다른 절반동안에는 짝수를 내보내기때문에 규칙적인 수법으로 RandomOffset에 값주기된다. 만일 RAND_MAX의 값이 자주 바뀐다면 0은 RandomOffset에 할당될 때 증가되며 그것은 값 0 ~ RAND_MAX사이에 그 홀수보다 더 많은 수가 있기때문이다.

RAND_MAX의 값이 IntervalSize와 비교할 때 매우 크다면 일부 수들은 무시될수 있는 경우가 1/(RAND_MAX + 1)로 된다. 이렇게 하여 식 Rand() % IntervalSize는 류사한 수법으로 0부터 IntervalSize-1까지의 수값을 만든다.

RandomOffset의 값이 수값 0부터 IntervalSize까지의 규칙적인 란수들이므로 식 GetLow() +

RandomOffset의 값은 low부터 low + IntervalSize-1까지의 규칙적인 란수이다. IntervalSize는 값 High-low + 1와 같으므로 Draw()에 의해 number에 할당되여 귀환되는 값은 수값 Low부터 High값까지의 규칙적인 란수들이다.

```
int number = GetLow() + RandomOffset;
```

프로그람 8-2는 수값 10부터 15에서 5개의 수들을 생성하기 위하여 EzRandomize()와 Draw()의 리용을 서술하고 있다.

```
#include <iostream>
#include <string>
#include "randint.h"
using namespace std;
int main() {
    EzRandomize() ;
    RandomInt U(10, 15);
    for ( int i = 1; i <= 5; i++) {
        cout << U.Draw() << end1;
    }
    return 0;
}
```

프로그람 8-2. randint 서고의 실례

프로그람 8-2의 실행값은 다음과 같다.

```
15
11
10
14
14
```

프로그람을 다시 실행시키면 다른 값을 내보낸다.

```
12
14
10
13
11
```

붉은색-노란색-풀색유희는 이러한 ADT의 완성된 란수서고를 리용한다.

콤퓨터의 력사

《백설공주와 7명의 난쟁이들》

콤퓨터로써 눈부신 성과를 거둔 회사의 하나가 Remington Rand회사였다. ENIAC의 설계자들인 모클

리(Mauchly)와 에커트(Eckert)는 만능자동콤퓨터 즉 UNIVAC라고 하는 Remington Rand의 콤퓨터완성품을 내놓았다. UNIVAC는 1952년 대통령선거의 결과를 예측하는데 CBS에다 방조를 주어 모두의 눈길을 모았다. 이 텔레비죤전시회로 하여 UNIVAC는 콤퓨터의 동의어로 되였다. 비록 콤퓨터기술의 개발이 새로운 회사들에 의해 진행되였다고 해도 그때 회사들은 그 마당에 느리게 들어 서고 있었다.

1952년에 Thomas J,Watson,Jr에 의해 IBM은 자기회사에 콤퓨터생산기지를 추가하였다. 처음으로 생산된것은 IBM 701이였다. 1964년에 IBM System 360이 도입되여 IBM은 이 산업에서 주도권을 확고히 틀어 쥐였다. 이 기간에 업무콤퓨터시장에서의 IBM의 지배가 대단히 큰것으로 하여 콤퓨터산업을 흔히 《백설공주와 7명의 난쟁이들》라고 불렀다. 여기서 《백설공주》는 IBM이며 《7명의 난쟁이들》은 Sperry Rand,Control Data,Honeywell,RCA,NCR, General Electric,Burroughs인데 이 회사들은 대형콤퓨터들을 생산하지 못하였다.

8.7 붉은색-노란색-풀색유희

붉은색-노란색-풀색유희는 추측유희이다. 그것은 보통 두 사람중 한 사람이 100과 999사이에서 하나의 수를 택하면 다른 사람이 그것을 추측하는것으로 진행한다. 추측이 진행될 때마다 제시자(수를 택한 사람)는 추측자에게 추측의 붉은 수자, 노란 수자, 풀색 수자의 개수를 말하는 방법으로 응답한다.

추측수자가 대응하는 대답수자와 같으면 풀색수자로 된다. 추측수자가 대답수자의 그 어느것에도 대응되지 않으면 붉은 수자로 된다. 붉은색도 풀색수자도 아니면 노란색수자이다. 대답이 123이고 추측이 422이라고 하자. 대답은 붉은색 하나, 노란색 하나, 풀색 하나로 된다. 첫 수자가 4, 붉은색 수자이다. 왜냐하면 그것이 1, 2, 3도 아니기때문이다. 가운데수자 2는 풀색수자인데 그것은 대답수자에 맞았기때문이다. 마지막수자 2는 노란색수자이다. 왜냐하면 대답수자는 맞히지 못했지만 대답수자의 다른것과는 맞았기때문이다(가운데대답수자).

알아맞추기에서 응답하는데 붉은색-노란색-풀색정보를 리용하면 그 대답은 붉은색수자가 처음, 노란색수자가 다음, 그리고 그다음 풀색수자가 주어 진다. 총계가 주어 지기만 하면 유희는 더 힘들어 진다. 대답이 653이고 알아 맞춘것이 616이라고 하면 판정자는 붉은색 하나, 노란색 하나, 풀색 하나 라고 대답한다. 대신 대답이 492이고 알아 맞춘것이 249라면 판정자는 붉은색 0, 노란색 3, 풀색 0이라고 대답한다.

8.7.1 추상화와 대면부

목표는 수제시자의 역할을 수행하는 프로그람유희를 개발하는것이다.

객체지향설계에서 초기단계는 체계를 구성하는 객체들을 결정하는것이다. 이 경우에 체계는 붉은색-노란색-풀색유희이다. 총체적으로 유희를 조종하는 유희조종자가 있어야 한다. 또한 사용자의 입력과 출력을 표시하는 객체들이 있어야 한다. 그리하여 조종자와 결합된 객체들은 현재 사용자가 알아 맞힌 세개의 수자와 그 추측에 대응하는 세개의 색갈을 표현하여야 한다. 조종자와 결합된 다른 객체는 프로그람에 의하여 수립된 3개의 수자여야 한다. 이 조종자의 모든 객체들이 3개의 수자적인 부분품을 가진다는데 주의하시오. 이렇게 되면 3개의 수자적인 부분품들을 가지는 객체를 표현하는 클라스를 개발하는것이 유리한다. 클라스와 같은 객체들이 알아 맞추고 응답하며 판정된 수자들을 표시하는데 리용될수 있다.

객체지향설계에서 다음단계는 객체들이 어떻게 호상작용하는가 하는것이다. 체계에서 조종자는 붉은색-노란색-풀색응답을 발생시켜 사용자알아맞추기에 반응할수 있어야 한다. 그러한 응답은 다음사용자의 알아맞추기를 발생한다. 이러한 호상작용은 사용자가 수자를 정확하게 알아 맞히거나 포기할 때까지 계

속한다. 여러가지 객체들의 동작을 더 세밀하게 고찰해 보자.

추측객체의 동작

- 알아맞추기를 진행한다.
- 하나의 추측에 하나의 부분수자로 값을 값주기하게 한다.
- 추측에서 부분수자의 값을 검사한다.

응답객체의 동작

- 추측에 대응하여 응답을 초기화한다.
- 응답에 붉은색, 노란색, 풀색성원을 값주기한다.
- 응답에서 붉은색, 풀색, 노란색성원을 검사한다.

중요한 유희조종자객체의 동작

- 유희를 시작하기 위하여 100부터 999사이의 하나의 세자리수자를 자유로 선택한다.
- 사용자를 접수한다.
- 사용자가 추측을 하도록 한다.
- 그 추측을 얻는다.
- 응답을 생성하기 위하여 추측을 평가한다.
- 응답을 표시한다.
- 옳은 추측을 찾는다.
- 옳은 추측을 환영한다.
- 사용자가 탈퇴하도록 한다.
- 유희를 조절한다.

지원하는 유희조종자동작

- 주어 진 수에서 수자를 검사한다.
- 주어 진 수에 수자를 배치한다.
- 응답에 대한 붉은색, 노란색, 풀색수를 결정한다.

세자리수자를 가지는 객체동작

- 개별적인 부분을 검사한다.
- 개별적인 부분을 설정한다.

세자리수자(자료성원들)를 가지는 객체를 표현하는 클라스이름은 Element이다. Element에 대한 머리부파일은 목록 8-12에 주었다. 클라스는 기정으로 자기성원들을 0으로 초기화하는 하나의 구축자를 가진다. 앞에서 언급된 동작들을 수행하기 위한 세개의 public검토자와 변이자가 있다. 이 성원함수들은 정보은폐화를 제공하여 준다.

목록 8-12. Element클라스의 element.h 머리부파일

```
#ifndef ELEMENT_H
#define ELEMENT_H
class Element {
```

```
    public:
        Element(int x=0, int y=0, int z=0);
        Int  GetX() const;
        int GetY() const;
        int GetZ() const;
        void SetX(int x);
        void SetY(int y);
        void SetZ(int z);
    private:
        int x;
        int y;
        int z;
};
#endif
```

Element의 세개 private자료성원들은 x, y, z라고 이름 지었다. 이러한 이름들은 Element의 객체가 3차원공간자리표와 비슷하기때문에 선택되였으며 거기에서 축들은 x축, y축, z축이라고 부른다. 자료성원에 대한 이름은 검토자와 변이자에 의해 규정된다. 검토자와 변이자가 결합되는 이름들은 즉시에 뒤따른다.

목록 8-13. Guess클라스의 guess.h머리부파일

```
#ifndef GUESS_H
#define GUESS_H
#include <iostream>
#include <string>
#include "element.h"
using namespace std;
class Guess {
    public:
        //기정구축자
        Guess();
        //검토자
        int GetDigit(int I) const;
        //변이자
        bool Update();
    protected:
        //변이자
        void SetDigit(int I, int v);
    private:
```

```
        //원소의 개수
        Element Number;
};
//보조연산자
ostream& operator<<(ostream &sout, const Guess &G);
#endif
```

사용자알아맞추기객체를 표현하는 클라스이름은 Guess이다. Guess에 관한 머리부파일을 목록 8-13에 주었다.

객체동작을 언급하는데 따라 Guess에 관한 클라스정의에서는 성원함수들이 주어 진다. public검토자 GetDigit()는 사용자추측에서 특수한 수자에로 호출을 지적한다. 변이자 UpDate()는 다음추측을 얻는다. 8.7.2에서 준 과정의 완성에서 UpDate()는 표준입력흐름으로부터 사용자추측을 입력한다. 다른 완성은 또한 UpDate()를 간단히 수정하여 진행할수 있다. Protected변이자 SetDigit()는 추측표현에서 개별적인 수자들을 설정하는데 리용된다. private자료성원 number는 Element형에서 3개의 수자들을 기억한다. 추측에서 첫 수자는 number의 x성원과 결합되며 두번째 성원은 y성원, 세번째 수자는 z성원과 각각 결합된다. 머리부파일은 또한 Guess객체에 대한 다중정의된 입력연산자의 기본형을 포함하고 있다.

Response는 사용자에게 대답을 주는 클라스이름이다. Reponse에 관한 머리부파일은 목록 8-14에 있다. 객체동작의 언급에 따라 성원함수들이 다시 클라스선언에서 주어 진다. public검토자는 붉은색, 노란색, 풀색성원에로의 호출을 진행한다. 구축자는 응답에서 붉은색, 노란색, 풀색성원들에 대한 총계를 설정하기 위하여 protected변이자를 리용한다. 이 변이자는 일단 응답이 구축되면 수정할 필요가 없으므로 protected부분안에 있다. 색수는 Element의 private자료성원 counts를 리용하여 표시한다. 여기에서 붉은색수는 counts의 x성원과 결합되며 노란색수는 y성원, 풀색수는 z성원과 결합된다.

목록 8-14. Response클라스의 response.h머리부파일

```
#ifndef RESPONSE_H
#define RESPONSE_H
#include "element.h"
class Response {
    public:
        Response(int r = 0, int y = 0, int g = 0);
        int GetRed() const;
        int GetYellow() const;
        int GetGreen() const;
    protected:
        void SetRed( int r);
        void SetYellow(int y);
        void SetGreen(int g);
    private:
```

```
        Element Counts;
};
#endif;
```

유희를 관리하는 클라스이름은 RYG이다. RYG에 대한 머리부파일은 목록 8-15에 있다. 보조연산자를 가지고 클라스 Element , Guess, Response와 RYG는 함께 붉은색-노란색-풀색유희자료추상화를 형성한다.

목록 8-15. RYG클라스의 ryg.h머리부파일

```
#ifndef RYG_H
#define RYG_H
#include "guess.h"
#include "response.h"
#include "element.h"
class RYG {
    public:
        RYG();
        void play();
    protected:
        void Welcome() const;
        void Prompt() const;
        Response Evaluate(const Guess &C) const;
        void Congratulations() const;
        void Display(const Guess &G, const Response &R)  const;
        void GoodBye() const;
        int GetRed(const Guess &C) const;
        int GetYellow(const Guess &G) const;
        int GetGreen(const Guess &G) const;
    private:
        Element SecretNumber;
        Guess UserInput;
        Response UserFeedback;
};
#endif
```

Guess, Response, Element의 머리부파일은 자료성원 UserInput. User Feedback, SecrectNumber이기때문에 RYG에 대한 머리부파일에 포함된다. 두개의 **public**성원이 있다. 하나의 **public**성원은 Element객체 SecretNumber를 초기화하는 기정구축자인데 그것은 프로그람에 의하여 확정된 수자를 표시한다. 다른 **public**성원은 play()인데 그것은 유희조종을 관리한다. RYG의 다른 성원

들을 파악하기 위한 play()의 수행은 목록 8-16에 제공하였다.

목록 8-16. ryg.cpp의 RYG성원함수 play()

```
void RYG::Play() {
    Welcome();
    Prompt();
    while (UserInput.Update()) {
        UserFeedback = Evaluate(UserInput);
        Display(UserInput, UserFeedback);
        if (Winner(UserFeedback)) {
            Congratulations();
            return;
        }
        else {
            Prompt();
        }
    }
    GoodBye();
}
```

성원함수 play()는 성원함수 Welcome()과 Prompt()를 호출하는것으로부터 시작한다.

```
Welcome();
Prompt();
```

성원함수 Welcome()은 환영통보문을 표시하며 Prompt()는 첫번째 알아맞히기를 요구한다. 그다음 함수 Play()는 While순환에 들어 간다.

```
while (UserInput. Update()) {
    UserFeedback = Evaluate(UserInput);
    Display(UserInput, UserFeedback);
    if (Winner(UserFeedback)) {
        Congratulations();
        return;
    }
    else {
        Prompt();
    }
}
```

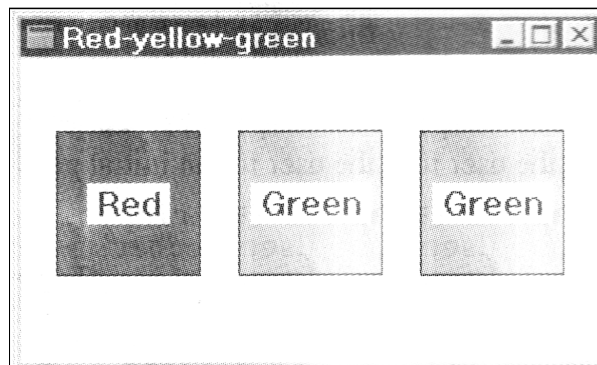
While순환은 사용자의 추측을 갱신하는 식(UserInput.Update())을 시험한다. 이 순환은 매 추측에 대하여 한번씩 진행된다. 만일 검사식이 틀린다면 사용자는 값을 제공하지 못하였으며 대신 주었다. 이 조건하에서는 순환이 탈퇴되고 적당한 통보문이 성원함수 GoodBye()에 의해 표시된다(그것은 사용

자가 유희에서 이기면 프로그람이 순환본체안으로부터 귀환명령문을 실행하는 경우일것이다).

GoodBye();

성원함수 Evaluate()는 적당한 응답을 결정하여 주기 위하여 사용자추측을 조사해 본다. 성원함수 Display는 본문과 도형방식으로 추측결과를 현시한다. 성원함수 Winner()는 응답을 조사하고 그것이 3개의 풀색을 지적하는가, 안하는가를 결정한다. 만일 성공응답이라면 성원함수 Congratulations()는 이 사실을 지적하고 함수 play()가 복귀한다. 응답이 성공이 아니라면 Prompt()는 순환의 다른 과정을 준비하기 위하여 실행된다. 이 시점에서 **While**순환은 추측을 갱신하려는 식을 시험하며 성공이라면 순환본체는 다시 실행된다. 함수 play()의 출력실례는 다음과 같다.

```
Welcome to the red - yellow - green game.
A number Between 0 and 999 has been chosen
for you to guess.

What is your guess? 456
Guess 456 corresponds to 1 red, 0 yellow, and 2 green

What is your guess? 536
Guess 536 corresponds to 1 red, 0 yellow, and 2 green

What is your guess? 526
Guess 526 corresponds to 1 red, 0 yellow, and 2 green

What is your guess? 596
Guess 596 corresponds to 0 red, 0 yellow, and 3 green

Congratulations on your win!
```

526에 대한 추측결과는 다음과 같다.

성원함수 play()는 함수 ApiMain()으로부터 호출된다. 목록 8-17에서 본것처럼 함수 ApiMain()은 두개의 명령 즉 유희구체례를 창조하기 위한 RYG객체 Game성원과 그 구체례에 대한 play()의 호출로 이루어 졌다. 성원함수 Evaluate()를 돕기 위하여 3개의 다른 성원함수(GetRed(), GetYellow(), GetGreen())들이 특별한 색갈과 관련한 추측을 분석한데 기초하여 정의된다.

목록 8-17. rygmain.cpp의 유희조종자

```
#include <iostream>
#include <string>
```

```
#include "ryg.h"
using namespace std;
int ApiMain() {
    RYG Game;
    Game.Play();
    return 0;
}
```

8.7.2 클라스 Element와 Guess의 실현부

여기서는 여러가지 클라스성원함수들과 보조연산자들의 실현부를 보여 준다. 클라스 Element부터 시작하자. 실현부를 목록 8-18에 주었다. 주어 진 3개의 검토자들 Element와 3개의 변이자들은 그 실현부에서 간단하다. 이 성원함수들은 정보은페를 제공하기 위하여 있는데 구체적인 서술은 하지 않는다.

목록 8-18. element.cpp의 Element클라스의 실현부

```
#include "element.h"
Element::Element(int x, int y, int z) {
    SetX(x);
    SetY(y);
    SetZ(z);
}
int Element::GetX() const {
    return X;
}
int Element::GetY() const {
    return Y;
}
int Element::GetZ() const {
    return Z;
}
void Element ::;SetX(int x) {
    X = x;
}
void Element::SetY(int y){
    Y = y;
}
void Element::SetZ(int z) {
    Z = z;
}
```

목록 8-19는 클라스 Guess의 실행과 Guess객체를 위하여 다중정의된 출력연산자를 포함하고 있다.

목록 8-19. guess.cpp에서 보조삽입연산자와 Guess성원함수의 실현부

```
#include <iostream>
#include <string>
#include <stdlib.h>
#include "guess.h"
using namespace std;
Guess::Guess() {
    Number.SetX(0);
    Number.SetY(0);
    Number.SetZ(0);
}
int Guess :: GetDihit(int I) const {
    int Digitvalue;
    switch (i) {
        case 1: Digitvalue = Number.GetX(); break;
        case 2: Digitvalue = Number.GetY(); break;
        case 3: Digitvalue = Number.GetZ(); break;
        default:
            cerr << "Bod digit request: " << i << end1;
            exit( 1 );
    }
    return digitvalue;
}
void Guess :: SetDigit(int i, int v) {
    switch (i) {
        case 1: Number.SetX(v); break;
        case 2: Number.SetY(v); break;
        case 3: Number.SetZ(v); break;
        defapit:
            cerr << "Bad digit request: " << i << end1;
            exit(1);
    }
}
bool Guess :: Update() {
    int Value;
    if (cin >> Value) {
        if ((Value >= 100) && (Value <= 999)) {
```

```
            int d1 = Value /100;
            int d2 = (Value % (d1 * 100)) / 10;
            int d3 = Value % 10;
            SetDigit(1, d1);
            SetDigit(2, d2);
            SetDigit(3, d3);
        }
        else
            cerr << "Illegal guess ignored." << end1;
        return true;
    }
    else
        return false;
}
ostream& operator <<(ostream &sout, const Guess &G) {
    sout << G.GetDigit(1) << G.GetDigit(2) << G.GetDigit(3);
    return sout;
}
```

Guess클라스구축자를 모두 0으로 만들기 위하여 자료성원 Number를 초기화한다.

```
Number.SetX(0);
Number.SetY(0);
Number.SetZ(0);
```

목록 8-19의 Guess검토자성원함수 GetDigit()는 사용자의 추측수자가 돌려 지는것을 결정하기 위하여 파라메터 i를 리용한다. 앞에서 언급한것처럼 첫 수자는 Number의 X성원에 보관되며 둘째 수자는 Y성원, 셋째수자는 Z성원에 보관된다. 검토자는 틀린 값을 결정하는 Number의 적당한 성원함수를 호출하기 위하여 switch명령문을 리용한다. 그 값은 Digitvalue에 할당된다.

```
switch (i) {
    case 1: Digitvalue = Number.GetX(); break;
    case 2: Digitvalue = Number.GetY(); break;
    case 3: Digitvalue = Number.GetZ(); break;
    default:
        cerr << "Unexpected digit request:" << i <<end1;
        exit( 1 );
}
```

switch명령문에서 default인 경우는 파라메터 i가 정확한 수자위치에 대응되지 않을 때이다. 이 경우에는 오유통보가 생성되고 프로그람이 끝난다. switch명령문이 끝나면 함수는 함수값을 생성하는 귀환명령문을 실행한다.

```
return DigitValue;
```

목록 8-19의 Guess변이자성원함수 SetDigit()는 i번째 수자를 설정하기 위하여 파라메터 i와 V를 리용한다. GetDigit(), SetDigit()로 호출되는 Number의 변이자성원함수를 결정하기 위하여 **switch**명령문을 리용한다. 완성되면 SetDigit()는 지적된 수자에 대하여 V가 유효한가를 검사하지 않는다. V의 유효성에 대해서는 이 장의 마지막에 련습으로 주었다.

목록 8-19의 Guess변이자성원함수 Update()의 과제는 다음사용자의 입력을 얻는것이다. 실행에서 사용자입력은 옹근수 value로 얻어 지며 Element형으로 변환된다. 추출이 정확하다면 시험에서는 값이 맞는가를 확인한다(즉 value는 100부터 999사이에 놓인다). value가 틀린다면 현재추측은 갱신되지 않는다. value가 옳다면 다음과 같은 기능이 주어 진다.

```
int d1 = Value / 100;
int d2 = (Value - (d1 * 100) /10;
int d3 = Value % 10;
```

객체 d1은 value의 값을 가리키며 value를 100으로 나누어서 구한다. 객체 d2는 value의 중간이며 d1에 100배 한 다음 value에서 덜어서 10으로 나누어서 얻는다.

객체 d3은 value를 10으로 나눈 나머지이다. 일단 추측한 수자들이 판정되면 변이자 SetDigit1()가 추측한 첫번째, 두번째 그리고 세번째 수자들을 새값으로 설정하기 위하여 호출된다.

```
SetDigit(1, d1);
SetDigit(2, d2);
SetDigit(3, d3);
```

추출이 진행된 후 값 **true**가 돌려 진다. 이 값은 사용자가 다른 추측을 제공하였다는것을 지적한다. 만일 value추출이 불가능하다면(즉 식 cin >> value 가 오유로 평가된다.) UpDate()는 **false**로 돌린다.

목록 8-19에서 **false**식에 대하여 삽입연산자 <<의 동작은 2개의 명령문으로 이루어 진다. 첫번째 명령은 실제적인 삽입을 세 수자들로 따로따로 표시하기 위하여 G의 성원함수 GetDigit()를 리용하여 수행된다. 다른 명령문은 흐름 sout의 참조귀환이다. National객체에 대한 출력연산의 다중정의로서 참조귀환은 Guess출력과 더 큰 출력명령문의 부분으로 된다. 실례로 다음과 같은 코드부분에서 Guess객체 MyGuess가 처음에 표시되고 그다음에 행바꾸기문자가 표시된다.

```
Guess MyGuess;
cout << MyGuess << "\n";
```

8.7.3 클라스 Response의 실현부

클라스 Response의 완성을 목록 8-20에 주었다. 이 Response구축자는 응답과 련결된 3개의 자리표들을 초기화하기 위하여 파라메터 r, y, g를 리용한다(파라메터들은 기정값이 0이다). 그 자리표들은 자료성원 Element객체 counts에 보관된다. 초기화는 변이자 SetRed(), SetYellow(), SetGreen()를 리용하여 수행된다.

```
SetRed( r);
SetYellow(y);
SetGreen(g);
```

세 파라메터들의 값이 정확한가 하는 확인은 이 장의 마지막에서 련습으로 준다.

앞에서 언급한것처럼 붉은색수는 Element counts의 x성원과 결합되며 노란색수는 y성원, 풀색수는 z성원과 결합된다. Response검토자와 변이자는 정보은페를 지원한다. 그것들의 수행은 간단하다.

목록 8-20. response.cpp에서 Response성원함수의 실현부

```
#include "response.h"
//Response():
Response::Response(int r, int y, int g) {
    SetRed(r);
    SetYellow(y);
    SetGreen(g);
}
//GetRed():붉은색수자를 얻는다.
int Response::GetRed() const {
    return Counts.GetX();
}
//GetYellow():노란색수자를 얻는다.
int Response::GetYellow() const {
    return Counts.GetY();
}
//GetGreen():풀색수자를 얻는다.
int Response::GetGreen() const {
    return Counts.GetZ();
}
//SetRed():붉은색수자를 설정한다.
void Response::SetRed() const {
    Counts.SetX( r);
}
//SetYellow():노란색수자를 설정한다.
void Response::SetYellow() const {
    Counts.SetY(y);
}
//SetGreen():풀색수자를 설정한다.
void Response::SetGreen() const {
    Counts.SetZ(g);
}
```

8.7.4 클라스 RYG의 실현부

이제 RYG성원함수들의 실현부를 고찰하자. 그의 완성은 목록 8-21부터 8-23까지 그리고 이전의 목록 8-16에서의 성원함수 Play()를 포함한다.

목록 8-21에는 여러가지 서고들이 포함되며 그것들중의 하나가 이 장에서 언급하였던 란수서고이다. 이 목록은 또한 SimpleWindow객체 Wout를 정의한다. 이 객체는 EzWindow도형방식의 객체들이 그려 지는 창문이다. 선택된 방법은 전역객체로 하기보다는 RYG자료성원으로서 이 객체를 가지도록 하는것이다. 그러나 성원과 같은것의 초기화는 C++의 성원초기화목록기구의 리용을 요구하는데 그것에 대한 언급은 다음장에까지도 없다.

기정구축자는 목록 8-21에서 정의된 첫 성원함수이다. 구축자는 도형방식창문 wout를 열고 매번 유희이 실행될 때 같은 수자들이 얻어 지지 않게 하는 EzRandomize()를 호출하는것으로 시작한다.

목록 8-21.　　　　　　ryg.cpp에서 RYG성원함수의 실현부

```
#include <iostream>
#include <string>
#include <stdlib.h>
#include "randint.h"
#include "rect.h"
#include "label.h"
#include "ryg.h"
using namespace std;
SimpleWindow wout("Red-yellow-green", 10, 4);
//기정 RYG구축자
RYG::RYG() {
    wout.Open();
    EzRandomize();
    RandomInt x(1, 9);
    RandomInt y(0, 9);
    RandomInt z(0, 9);
    SecretNumber.SetX(x.Draw());
    SecretNumber.SetY(y.Draw());
    SecretNumber.SetZ(z.Draw());
}
// Welcome():열려 진 대화통보현시
    void RYG::Welcome() const {
        cout << "Welcome to the red-yellow-green game.\n"
        << "A number between 100 and 999 has been chosen\n"
        << "for you to guess.\n" << end1;
}
```

```
// Prompt():다음추측을 요구한다.
void RYG::Prompt() const {
    cout << "What is your guess? ";
}
// Congratulations(): 성공을 알린다.
void RYG::Congratulations() const {
    Display(UserInput , UserFeedback);
    cout << "Congratulations on your win!" << end1;
}
//GoodBye(): 사용자에게 감사통보를 알린다.
void RYG::GoodBye() const {
    cout << "Better luck next time" << end1;
}
```

```
wout.Open();
EzRandomize();
```

기정구축자는 다음 RandomInt객체 X, Y, Z를 정의한다. 이 객체들은 프로그람에 의해 주어 지는 수인 3개의 수자들을 생성하는데 리용된다.

```
RandomInt x(1, 9);
RandomInt y(0, 9);
RandomInt z(0, 9);
```

객체 X는 첫 수자를 생성하는데 리용된다. 수가 100부터 999까지의 사이에 놓이기때문에 수자로서 표시되는 모조란수값은 1부터 9까지에 놓이게 된다. 객체 Y와 Z는 두번째와 세번째 수자들에 대한 모조란수값을 나타내는데 리용된다. 이 두개의 수자들은 그 값에서 제한이 없으며 그래서 Y와 Z는 0부터 9까지의 수자로 된다.

다음 RYG구축자는 자기의 Element자료성원 SecretNumber의 세부분들을 구성한다. 앞에서 언급한것처럼 SecretNumber의 X성원은 프로그람에 의해 주어 진 수의 첫 수자와 결합되며 Y성원은 두번째 수자, Z성원은 세번째 수자와 결합된다. SecretNumber의 변이자는 파라메터로서 적당한 간격으로 표현된 모조란수를 리용한다.

```
Secretnumber.SetX(x.Draw());
Secretnumber.SetY(y.Draw());
Secretnumber.SetZ(z.Draw());
```

목록 8-21의 다른 성원함수들 Welcome(), Prompt(), Congratulations(), GoodBye()는 간단하므로 분석할 필요가 없다. 목록 8-22의 RYG성원함수 Winner()는 파라메터 r가 성공대답을 표현하는가, 아닌가를 결정한다. 파라메터 r는 그와 련결되는 풀색수가 3이라면 성공대답이다. R의 성원함수 GetGreen은 실제적인 풀색수를 계산한다.

목록 8-22에 준 다른 RYG성원함수는 Display이다. 이 함수는 두개의 파라메터를 가지는데 그것은 Guess G와 Response R이다. Display()성원함수는 추측의 결과를 표시한다..

Display의 실현부는 r가 결합되는 자리표들로 국부객체 rcount, ycount, zcount를 초기화하는 r의 검토자호출로부터 시작한다.

```
int rcount = R.GetRed();
int ycount = R.GetYellow();
int gcount = R.GetGreen();
```

이 자리표값들은 본문방식응답을 생성하는 출력명령들에서 리용되게 되며 도형방식출력을 생성하는 for순환을 조종하는데 리용되게 된다. 첫 출력명령은 추측에 관한 정보를 생성한다.

```
cout << "Guess " << G << "corresponds to";
```

출력명령이 Guess객체를 G로 표시하는 <<연산자의 다중정의를 리용한다는데 주의하시오. 다음과 같은 출력명령은 사용자에게 3개의 결과값들을 보여 준다.

```
cout << rcount << "red, " << ycount << "yellow, and"
  << gcount << "green" << "\n"<<end1;
```

도형방식의 표식은 다음에 생성된다. 이 기능을 위하여 객체 cx와 cy가 정의된다. 그것들은 표시할 다음 색의 중심자리표를 표시한다.

목록 8-22. ryg.cpp에서 RYG성원함수의 실현부

```
bool RYG::Winner(const Response &R) const {
    return R.GetGreen() == 3;
}
void RYG::Display(const Guess &G, const Response &R) const {
    int rcount = R.GetRed();
    int ycount = R.GetYellow();
    int gcount = R.GetGreen();
    cout << "Guess " << G << "corresponds to";
    cout << rcount << "red, " << ycount << "yellow, and"
      << gcount << "green" << "\n"<<end1;
    float cx = 2;
    float cy = 2;
    for(int r = 0; r < rcount; ++r) {
        RectangleShape Box(wout, cx, cy, Red, 2, 2);
        Box.Draw();
        Label S(wout, cx, cy, "Red");
        S.Draw();
        cx +=3;
    }
    for(int y = 0; y < rcount; ++y) {
        RectangleShape Box(wout, cx, cy, Yellow, 2, 2);
```

```
        Box.Draw();
        Label S(wout, cx, cy, "Yellow");
        S.Draw();
        cx +=3;
    }
    for(int g = 0; g < rcount; ++g) {
        RectangleShape Box(wout, cx, cy, Green, 2, 2);
        Box.Draw();
        Label S(wout, cx, cy, "Green");
        S.Draw();
        cx +=3;
    }
}
```

객체 cx와 객체 cy는 둘 다 초기에는 2이다. 순환에서 cx는 3만큼 증가되는데 통의 한변의 길이의 1.5배이다.

```
float cx = 2;
float cy = 2;
```

순환이 매우 반복될 때마다 RectangleShape객체가 정의되고 표현된다. 류사하게 RectangleShape 객체의 색갈을 지적하는 표식이 생성된다. 실례로 붉은색통을 표시하는 순환을 참조하시오.

```
for(int r = 0; r < rcount; ++r) {
    RectangleShape Box(wout, cx, cy, Red, 2, 2);
    Box.Draw();
    Label S(wout, cx, cy, "Red");
    S.Draw();
    Cx += 3;
}
```

순환은 자리표 (cx, cy)에 위치하는 적당한 색갈과 크기의 객체 box를 구축하는것으로 시작한다. 객체를 표현한후에 통의 중심에 나타나는 표식이 구축되고 다음에는 표현된다. 그다음 통의 중심이 현재 통의 오른쪽에로 1.5단위 떨어 져 있기때문에 cx는 3만큼 증가된다. 그렇게 되면 다른 붉은색통이 표현될 필요가 있는가, 없는가를 시험해 볼수 있다. 다른 두개의 순환들도 비슷한 방식으로 연산한다.

목록 8-23은 RYG촉진자함수 GetGreen(), GetRed(), GetYellow(), Evaluate ()의 정의이다. 모든 4개의 촉진자함수들은 Guess형의 가상파라메터 G를 받는다.

목록 8-23. ryg.cpp에서 일부 다른 RYG성원함수의 실현부

```
int RYG::GetGreen(const Guess &G) const {
    int green = 0;
```

```
    if (G.GetDigit(1) == SecretNumber.GetX())
        ++green;
    if (G.GetDigit(2) == SecretNumber.GetY())
        ++green;
    if (G.GetDigit(3) == SecretNumber.GetZ())
        ++green;
    return green;
}
int RYG::GetRed(const Guess &G) const {
    int sx = SecretNumber.GetX();
    int sy = SecretNumber.GetY();
    int sz = SecretNumber.GetZ();
    int gx = G.GetDigit(1);
    int gy = G.GetDigit(2);
    int gz = G.GetDigit(3);
    int red = 0;
    if ((gx != sx) && (gx != sy) && (gx != sz))
        ++red;
    if ((gy != sx) && (gx != sy) && (gy != sz))
        ++red;
    if ((gz != sx) && (gx != sy) && (gz != sz))
        ++red;
     return red;
}
int RYG::GetYellow(const Guess &G) const {
    return 3 - GetGreen (G) - GetRed(G);
}
Response RYG::Evaluate(const Guess &G) const {
    return Response(GetRed(G), GetYellow(G), GetGreen(G));
}
```

촉진자 GetGreen()은 G와 관련된 풀색수를 표현한다. 이 값은 국부적인 **int**객체 Green에 보관되는데 그것은 0으로 초기화된다. 함수 GetGreen()은 프로그람에 의해 주어 진 수의 첫번째 수자와 첫 추측수자를 비교한다. 첫 추측수자의 값은 G의 검토자 GetDigit()에 의하여 얻어 진다. 프로그람에 의해 주어 진 수의 첫 수자는 SecrectNumber의 검토자 GetX()를 통하여 얻어 진다.

```
if (G.GetDigit(1) == SecretNumber.GetX())
    ++green;
```

수자들이 같다면 풀색수자의 총계가 하나만큼 증가된다. 만일 맞지 않는다면 특별한 기능을 주지 않

는다. 다음 그 함수는 프로그람에 의해 주어 진 수의 두번째 수자와 추측의 두번째 수자를 비교해 보고 그것들이 같다면 풀색수자의 총계를 증가시킨다. 마지막으로 검토자는 프로그람에 의해 주어 진 세번째 수자와 추측의 세번째 수자를 비교하고 같다면 풀색수를 증가시킨다. 만일 비교와 증가가 완성되면 풀색수자들의 총계가 귀환되게 된다.

주의

안전한 입력추출

전문적인 쏘프트웨어는 일반적으로 입력을 문자표현으로 추출한다. 그때 이 입력은 유효하며 요구되는 표현으로 번역된다. 실례로 붉은색-노란색-풀색유희에서 3개의 char변수로서 추측을 추출할수 있다. 이 방법은 추측추출의 안전한 방식을 제공한다. 실례로 int표현이 리용되고 사용자가 수자가 아닌것을 입력하였다면 프로그람은 대부분의 사용자들에게 리해할수 없다는 오유통보문을 내보내고 완료한다. char형표현을 리용하면 Guess는 자기의 기능을 수행할수 있다. 련습에서 ADT에 대한 수정을 참조하시오.

GetRed()의 기능은 GetGreen()과 류사하다. 처음 총계는 0으로 초기화되며 프로그람에 의해 주어 진 수의 수자들과 추측수자들을 비교하고 그것들이 같으면 총계가 하나 증가되며 그 과정은 세번의 비교가 끝난 다음 귀환된다. 이 과제들을 수행하는데서 표를 안정하게 하기 위하여 함수 GetRed()는 추측된 수자들과 프로그람에 의해 주어 지는 수의 수자들이 복사를 진행한다.

```
int sx = SecretNumber.GetX();
int sx = SecretNumber.GetX();
int sz = SecretNumber.GetZ();
int gx = G.GetDigit(1);
int gy = G.GetDigit(2);
int gz = G.GetDigit(3);
```

추측수가 붉은색이라면 그것은 프로그람에 의해 주어 진 수의 수자들과 맞지 않는다는 의미이다. 3개의 다른 색이 요구되므로 추측수자가 붉은색수자인가, 아닌가 하는것을 결정하는 비교표현이 함께 결합되는 3개의 항목을 가진다(즉 &&연산자로 결합된다). 개별적인 항목은 추측수자가 현재프로그람에 의해 주어 진 참조되는 수자와 다르다면 참이다. 만일 모든 3개의 항목들이 참이라면 추측수자는 붉은색수자 이다. 실례로 첫 추측수자 gx는 그림 8-2에서 표현이 참이면 붉은색수자이다.

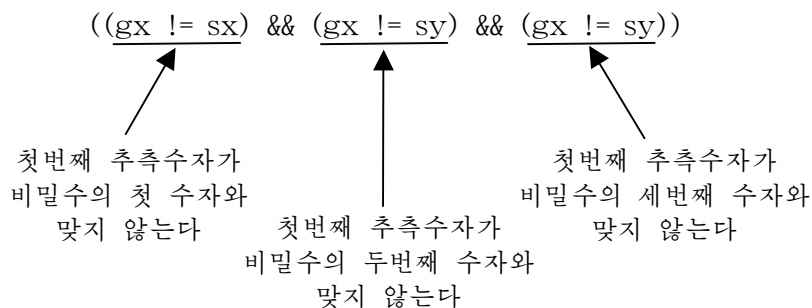



그림 8-2. 추측자가 붉은색수자인가를 검사하는 식

목록 8-23의 성원함수 GetRed(), GetGreen(), GetYellow()의 구체례를 얻는것은 간단하다. 함수 GetYello()는 간단히 3에서 붉은색과 풀색수를 던것을 귀환시킨다.

```
return 3 - GetGreen( G) - GetRed(G);
```

함수 Evaluate()도 목록 8-23에 주었다. 함수는 Guess G에 대응한 Response객체를 구축하고 귀환시킨다. Response객체는 Response구축자의 실제적인 파라메터로서 GetRed(G), GetYellow(G), GetGreen(G)의 값을 리용하여 구축된다.

```
return Response(GetRed(G), GetYellow(G), GetGreen(G));
```

이것으로써 RYG ADT의 언급과 ADT들에 대한 전면적인 개괄은 끝마친다.

8.8 알아 둘 점

- ✓ 수행되여야 할 정보와 연산에 대한 표현이 자료추상화이다.
- ✓ 추상자료형 혹은 ADT는 정보은페화원리를 리용하는 잘 정의되고 완성된 자료추상화이다. ADT는 자연스로운 방법으로 객체를 창조하고 조종하도록 한다.
- ✓ ADT최소화규칙: 비록 동작이 일반적으로 요구된다고 해도 그것은 ADT부분으로 될수 없다는것이다.
- ✓ 클라스최소화원리: 만일 함수나 연산자가 클라스성원이 아니지만 정의 될수 있다면 그것은 성원으로 될수 없다. 이것은 비성원함수나 연산자들이 클라스의 실행에 영향을 주지 않도록 한다.
- ✓ ADT에서 정보은페화를 실현함으로써 의뢰기프로그람들은 ADT의 실행변화로부터 영향을 받지 않는다.

콤퓨터의 력사

놀랄만한 매력

왜 콤퓨터하드웨어나 쏘프트웨어의 오유를 bug라고 하는지 알고 있는가? 단어 "bug"의 리용은 미해군중장 그레이스 호퍼 (1904-1987)의 공적으로 널리 인정되고 있다. 그레이스 호퍼(아래의 그림을 보시오)는 2차세계대전기간 해군의 Mark1콤퓨터에 대한 첫 프로그람작성자였다. 해군의 다른 콤퓨터들에서의 계산오유에 관한 원인추적에서 그 녀자는 그것이 계전스위치에 붙은 좀벌레에 의한 전자적인 단락에 기인된것이라는것을 발견하였다. 그때부터 오유수정을 벌레제거라고 부르게 되였다.

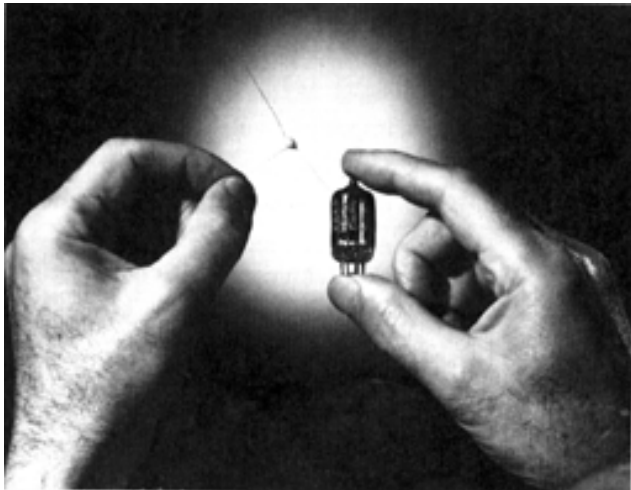
그레이스 호퍼는 초기에는 기술사업에 이바지하였다. 실례로 그 녀자는 UNIVAC의 설계자들중의 한 사람이다. 후에 그는 업무응용프로그람에 보다 더 적합한 콤퓨터환경의 필요성을 인식하고 BORMAC와 널리 리용되고 있는 COBOL을 개발하였다. 그레이스 호퍼는 실력 있는 수학자였다.

- ✓ C++에서 ADT는 클라스와 함수 그리고 연산자들을 리용하여 수행된다.
- ✓ 구축자는 ADT형의 객체들을 초기화한다. 그것은 매 ADT객체가 초기화된 모든 자료성원들을 가지는가를 확인하는 표준적인 실행이다.
- ✓ 구축자는 클라스와 같은 이름을 가진다.
- ✓ 기정구축자는 파라메터를 요구하지 않는 구축자이다.
- ✓ 복사구축자는 이미 정의된 원천객체의 복사로 되는 새로운 객체들을 초기화한다. 클라스가 복사구축자를 정의하지 않으면 콤파일러는 자동적으로 하나의 판본을 제공한다.

✓ 성원별복사는 원천객체들의 자료성원들이 목적객체에로 비트별로 복사되는 복사이다.

콤퓨터의 력사

반도체와 집적회로

1947년에 반도체가 AT&T Bell Telephone Laboratories 에서 발명되였다. 반도체는 본질적으로 진공관의 기능을 수행하지만 더 작고 간편하며 더 빠르고 전력소비가 적다. 그의 대비적인 크기를 아래의 그림에 보여 주었다.

콤퓨터들은 지난 1950년에 출현하였으며 2세대콤퓨터는 반도체로 설계되였다. 그것들이 만들어 진때로부터 반도체콤퓨터들은 진공관으로 만들어 진 콤퓨터보다 더 작아 지고 빨라 졌으며 간편하고 쓰기편리해졌다.

이전의 반도체로 만든 대부분의 콤퓨터들은 아주 컸으며 특별한 조종실과 그 장치를 가동시키기 위한 전력을 요구하는데 너무 엄청나는 에네르기랑비가 있었다. 어느 한 회사 Digital Equipment Corporation (DEC)이 소형콤퓨터를 만드는데 전문하였다. Digital 의 첫 소형콤퓨터 PDP-8은 1963년에 소개되였으며 서류통보다 좀 컸다. 그후 크기가 더 작아 졌다. 극소형콤퓨터의 유익한 점은 사용자가 임의의 곳에 이것들을 설치할수 있다는것이다. 구체적으로 말하면 그것들은 대형콤퓨터보다 덜 비싸며 연구비를 얻으려고 애쓰는 대학연구사들에 의해 만들어 졌다.

다른 기술적인 도약은 텍사스 인스트러먼트회사에서 일하는 잭크 킬비라는 기사에 의해 집적회로가 발명된 1958년에 일어 났다. 지난 1950년대와 1960년대의 반도체화된 콤퓨터는 선들로 결합되고 반도체소자를 땜질하여 만든 기판으로 이루어 졌다. 직접회로는 이 부분품들을 하나의 블로크, 보통 규소블로크상에 집적화할수 있게 한다. 집적회로를 리용한 첫 콤퓨터들은 1960중반기에 출현하였다. 그러나 집적회로의 가치는 1970년대전까지는 알수 없었다.

✓ 콤파일러에 의해 제공되는 복사구축자와 성원값주기연산자의 판본은 성원별복사를 수행한다.
✓ 해체자는 객체가 구체레 밖으로 나가면 클라스형 객체에 의하여 자 동적으로 호출되는 성원함수이다. 해체자는 해체된 객체에 대하여 필요한 지우기처리를 진행한다. 해체자의 이름은 기호 (~)과 클라스 이름의 결합으로 이루어 진다. 클라스는 하나의 해체자만을 가질수 있으며 그 해체자는 귀환값을 가지지 않는다.
✓ 공통적으로 존재하는 일부 다른성원함수들은 검토자, 변이자, 촉진자라고 한다.
✓ 검토자성원함수는 객체의 속성값을 돌린다.
✓ 검토자이름은 흔히 Get로 시작된다.
✓ 변이자성원함수는 객체의 속성을 수정하기 위한 방법을 제공한다. 변이자이름은 Set로 시작된다.
✓ 촉진자성원함수는 객체의 속성부분에 의거하여 과제를 수행한다.
✓ 함수대면부에 첨가되는 수식자 const는 함수가 그 어떤 자료성원도 수정하지 않는다는것을 가리킨다.

const성원함수는 클라스의 const객체들에 의해 리용될수 있다.

- ✓ 클라스객체에 대한 의뢰자대면부는 클라스정의의 public부분에서 발생한다.
- ✓ 임의의 부분(public, protected, private)에서 정의된 성원함수는 클라스의 다른 모든 성원들에 의해 호출될수 있다.
- ✓ protected부분의 성원들은 그 클라스의 다른 성원들에 의하여, 그 클라스로부터 파생된 클라스에 의하여 리용될수 있다.
- ✓ 자료성원들은 표준적으로 private부분에서 선언된다. ADT의 자료성원들에 대한 의뢰기프로그람이 직접 호출을 제한한다면 값의 완전성과 정확성을 확인하는데 도움이 된다.
- ✓ private성원은 오직 자기 클라스의 성원에 의해서만 리용될수 있다.
- ✓ 함수나 연산자에 대한 귀환형에서 &는 참조귀환이 수행된다는것을 의미한다. 참조귀환에서는 복사가 아니라 실제적인 객체에 대한 참조가 귀환된다. 귀환된 객체의 유효범위는 호출된 함수나 연산자에 대해 국부적이지 말아야 한다.
- ✓ 입출력흐름동작은 확장이 가능하다. 프로그람작성자정의형객체들도 출력과 삽입연산을 할수 있다.
- ✓ ADT서고들은 자주 ADT클라스로 되지는 않지만 객체들을 동작시키는 보조적인 함수들과 연산자들을 포함한다.
- ✓ 공통적으로 제공된 두 보조연산자들은 삽입과 출력이다. 그것들은 기본형객체들에 관하여 입출력과 같은 형식을 주는 성원함수로는 되지 않는다.
- ✓ 모조란수렬은 란수렬의 출현과 통계적속성을 가진다.

련습문제

8.1 매 클라스정의는 ADT를 정의하는가?

8.2 ADT에서 요구되는 속성은 무엇인가?

8.3 객체지향프로그람설계에서 정보은폐화원리가 왜 중요하게 나서는가?

8.4 자료성원과 성원함수사이의 차이점은 무엇인가?

8.5 변이자와 검토자성원함수사이의 차이점은 무엇인가?

8.6 성원과 보조함수들, 연산자들사이의 차이점은 무엇인가?

8.7 여러가지 접근지정자표식을 리용하는데서 차이점은 무엇인가?

8.8 수식자 const의 목적은 무엇인가?

8.9 구축자에서 왜 귀환형이 필요한가?

8.10 표준귀환형과 참조귀환형은 어떻게 차이나는가?

8.11 함수나 연산자의 리용에서 변경되지 않은 클라스객체들이 왜 값파라메터를 넘기지 않고 상수형참조파라메터로 입력되는가?

8.12 객체속성값을 되돌리는 성원함수는 무엇이라고 하는가?

8.13 객체의 속성값을 설정, 변경하는 성원함수를 무엇이라고 하는가?

8.14 정보은폐화를 리용하는 완성된 자료추상화는 무엇인가?

8.15 기정구축자는 어떤 파라메터를 요구하는가?

8.16 이전에 정의한 객체의 복사로서 새 객체를 초기화하는 구축자는 무엇인가?

8.17 클라스객체에 대한 의뢰자대면부는 클라스정의의 어떤 부분에서 발생하는가?

8.18 복사구축자와 성원값주기연산자들이 콤파일러에 의해 자동적으로 제공되여 어떻게 복사되는가를 설명하시오.

8.19 참조귀환은 어떻게 지적되는가? 참조귀환에서 어떤 종류의 객체들이 리용될수 있는가? 참조귀환에서 어떤 종류의 객체들이 리용될수 없는가?

8.20 자료성원들을 표준적으로 비공개부분에서 정의하는가?

8.21 클라스 RandomInt에 대한 구름표현을 개발하시오. 1부터 14까지의 값으로 된 RandomInt에 대한 클라스의 구체례를 보여 주시오.

8.22 클라스 RYG에 대한 구름표현을 개발하시오. 클라스의 구체례를 보여 주시오.

8.23 다음정의들의 결과는 무엇인가?

```
class Widget {
    public:
        bool Flag;
        Widget();
        Widget(int Value);
        int Getvalue() const;
    protected:
        int DataItem;
        void Setvalue(int Value);
};
```

1) 클라스 Widget는 몇개의 성원함수를 가지는가?
2) 클라스 Widget는 몇개의 자료성원을 가지는가?
3) 클라스 Widget에서 정의된 함수 Setvalue()는 구축자인가?
4) 함수 Setvalue()는 클라스 Widget의 공개성원함수인가?
5) Widget공개성원함수들은 Widget자료성원 dataItem을 호출할수 있는가?
6) 의뢰함수는 Widget자료성원 Flag를 호출할수 있는가?
7) 클라스 Widget는 정보은페화를 제공하는가?

8.24 클라스 A와 B 그리고 그것들의 부분적인 실현부를 고찰하시오. 3개의 다른 모듈에서 오유를 지적하고 설명하시오.

```
class A {
    public:
        A();
        A(&int n);
        int A(int n, int n);
        int A3;
    private:
```

```
        int A1;
        int A2;
        int A3;
};
class B {
    public:
        B();
        A myA1;
    private:
        A myA2;
};
B::B(int a1, int a2, int a3) {
    myA.A1 = a1;
    myA.A2 = a2;
    myA.A3 = a3;
    myA2 = A(a1+1, a2+1, a3+1);
}
```

8.25 두개의 공개구축자를 가지는 클라스 Vehicle을 고찰하시오. 하나는 파라메터가 없고 다른것은 옹근수파라메터 X를 가진다. 이 클라스는 파라메터로서 성원함수 evaluate()를 제공하며 Vehicle클라스의 코드에 의해서만 호출가능하다. 이 클라스는 론리형성원함수 Ischarged()를 제공한다. 임의의 다른 함수로부터 호출이 가능하다. 이 성원은 클라스의 임의의 성원들에 의해 수정되지 않는다. Vehicle클라스는 void set()성원함수를 제공하는데 그것은 옹근수파라메터 s를 가지며 vehicle클라스성원코드내에서만 호출가능하다. 옹근수자료성원 Myturbo은 vehicle의 객체에 의해서만 호출된다. 클라스 vehicle을 작성하시오.

8.26 달력을 표현하는 Date를 작성하시오. 클라스는 1752.9.14로 날자를 초기화하는 기정구축자를 제공한다. 다른 구축자는 월, 일, 년에 대응하는 3개의 옹근수파라메터를 리용하며 특수한 값으로 Date객체를 초기화한다. 클라스는 년, 월, 일을 호출하는 3개의 공개검토자와 변이자를 가진다. 연산자 ++와 --는 date객체에 정의되며 객체의 새값은 각각 다음날 , 이전 날을 다중정의한다. 덜기연산자는 두 날자들의 차이가 그사이의 날자로 되게 다중정의된다. Date객체들에 대하여 출력과 입력연산자를 다중정의한다. 생성된 입출력형태가 모두 정확한가 확인하고 그 결과는 출력에 나타난다. 또한 보조함수 Tostring()을 정의하는데 그것은 Date파라메터의 string판번호를 돌려 보내며 DayOfWeek()는 Date파라메터에 맞는 주사이의 날자를 돌려 보낸다. Dayofweek()의 귀환형은 일요일, 월요일, 화요일, 수요일, 목요일,금요일, 토요일이다. Dayofweek의 귀환형은 기호상수가 일요일, 화요일, 수요일, 목요일, 금요일, 토요일의 렬거형으로 될수 있다.

8.27 다른 산수, 유리수연산자에 대한 원형을 포함할수 있도록 유리수서고를 수정하시오.

8.28 덜기, 나누기를 진행하는 공개산수촉진자 subtract()와 divided()를 실행할수 있도록 유리수서고를 수정하시오.

8.29 보조유리수연산자 "-"와 "/"을 실행할수 있도록 유리수서고를 수정하시오.

8.30 유리수가 서로 소인 분자와 분모로 된 공개변이자 Reduce()를 실행할수 있도록 유리수서고를 변경하시오. 분자와 분모는 최대공약수에 의해 나누어 진다(최대공통약수를 구하기 위해서는 유클리드알고리듬을 리용하시오).

8.31 큰가 작은가를 검사하는 Equal()와 LessThan()의 공개유리수촉진자를 실행하도록 유리수서고를 수정하시오.

8.32 ==, <연산자를 실행하도록 유리수서고를 수정하시오.

8.33 <=, >, >=연산자를 실행할수 있게 유리수서고를 수정하시오. 련습 8.32에서 정의한 연산자를 리용하시오.

8.34 Rational의 성원함수 Extract()의 정의를 수정하여 분자와 분모에 "/"가 생기도록 변경하시오. Extract()함수가 검사될수 있는가와 SetDenominator()에 의해서도 검사될수 있는가를 알아 보시오.

8.35 rational.cpp에서 정의된 모든 성원함수들에 두개의 삽입명령문을 써넣으시오. 첫번째 삽입명령문은 명령문의 본체에 시작이 있어야 한다. 이 삽입은 성원이 인용된다는것을 의미한다. 두번째 삽입명령문은 명령문의 마지막에 놓인다. 이 삽입은 성원의 실행이 완료되였다는것을 의미한다. 프로그람 8-1을 실행하고 출력결과를 확인하시오.

8.36 Rational구축자와 Rational성원함수 Extract()는 변이자 SetNumerator()와 SetDenominator()를 리용한다. 이 성원함수들에 한개의 새로운 Rational성원 SetRational()을 리용하여 두개의 지령명령문을 써넣을수 있다. 이렇게 하여 두개로 리용할수 있는것처럼 할수 있다. SetRational()함수에는 **int**파라메터인 numer와 denom이 있다. 이 파라메터는 분자, 분모객체를 입력하는데 리용된다. Rational클라스정의에 새로운 성원형을 추가하여 rational.cpp 원천파일을 실행하시오. 새로운 **public**, **protected**, **private**성원이 만들어 졌는가를 알아 보시오.

8.37 복사구축자를 명백히 정의할수 있도록 유리수서고를 수정하시오.

8.38 객체에 류점수를 되돌리는 공개성원함수 floatingPoint()를 계산하는 유리수서고를 수정하시오.

8.39 cout의 기정파라메터의 값을 가지도록 rational성원함수 Insert()를 수정하시오. 기정값이 어디에서 지정되는가? 왜 그런가?

8.40 cin의 기정파라메터값을 가지도록 rational의 성원함수 Extract()를 수정하시오. 기정값이 어디에서 지정되는가? 왜 그런가?

8.41 파라메터 r와 n을 가진 보조함수 Power()를 설계하시오. 파라메터 r는 rational이고 n은 **int**이다. 이 함수는 r의 n제곱값을 돌려 보낸다. 파라메터와 귀환형을 정의하시오.

8.42 복소수를 취급하는 ADT서고 complex를 설계하시오. 보조함수와 연산자성원함수와 연산자를 정의하시오. 그리고 호출제한성을 알아 보시오. 표준복소수서고와 비교하시오.

8.43 련습 8.42의 복소수서고를 실행하시오. 사용자서고의 특성을 보여 주는 프로그람을 실행하시오.

8.44 목록 8-9의 C클라스는 공개검토자와 변이자성원함수가 다음의 보조삽입, 추출연산자를 추가할수 있게 한다.

```
string C::GetName() const {
    return name;
}
```

```
void C::SetName(const string &s) {
    name = s;
}
ostream& operator<<(ostream &sout, const (&c) {
    sout << c.GetName();
    return sout;
}
C operator+(const C &c, const C &d) {
    s = c.GetName() + d.GetName();
    C result;
    return result;
}
```

어떻게 main()함수가 다음의 결과를 출력하는가를 설명하시오.

```
int main() {
    S = "x";
    C x;
    S = "y";
    C y;
    S = "z";
    C z;
    cout << "displaying: " << x << endl;
    cout << "displaying: " << y << endl;
    cout << "displaying: " << z << endl;
    cout << "displaying: " << (x+y) << endl;
    cout << "displaying: " << (z+z+z) << endl;
    endl;
    return 0;
}
```

출력결과는 다음과 같다.

```
x: default constructed
y: default constructed
z: default constructed
displaying: x
displaying: y
displaying: z
xy: default constructed
```

```
xy: copy constructed using xy
displaying: xy
xy: destructed
zz: default constructed
zz: copy constructed using zz
zz: destructed
zzz: default constructed
zzz: destructed
displaying: zzz
zzz: destructed
zz: destructed
z: destructed
y: destructed
x: destructed
```

8.45 6장에서 본 증권가격구간문제에 대한 주간 증권정보를 주간 상품정보를 표현하는 ADT로 설계하고 개발하시오. ADT와 그것을 리용하여 증권가격구간문제를 푸는 프로그람을 실현하시오. 증권가격은 일반적으로 유리수로 되므로 증권정보를 Rational객체로 표현해야 한다.

8.46 압연로라를 모의하시오. 1부터 6사이의 모조란수값을 가지는 RandomInt객체인 Dice1과 Dice2를 정의하시오. 1만개의 압연로라를 모의하시오. 매 로라는 두수의 합을 계산한다. 이 합에 의하여 발생회수가 계산된다. 매합이 나타난 회수와 총합을 비교하시오.

8.47 수가 얻어 지는 곳에서 간격값이 부의 값을 가지면 RandomInt클라스가 정확히 동작할수 있는가?

8.48 tic-tac-toe유희의 구체례를 보여 주는 ADT를 설계하고 실행하시오. 질 좋은 도형을 현시하기 위해서 ADT는 EzWindows서고를 리용한다.

8.49 Guess클라스를 수정하여 3개의 char객체가 생기도록 하시오. 만일 얻어 진 객체가 정확한 형태라면 문자는 수자로 변환된다. 정확치 않다면 오유통보를 발생한다. 이런 추출방법의 우점은 무엇인가?

8.50 ADT객체가 추측자의 역할을 하도록 붉은색-노란색-풀색유희의 ADT를 설계하시오.

8.51 새 수자값이 정확한 값인가를 검사하도록 Guess성원함수 SetDigit()를 수정하시오.

8.52 파라메터가 정확한 색을 계산한다는것을 보여 주도록 Response구축자를 수정하시오(즉 파라메터 합은 3이고 개별적 파라메터는 1부터 3사이의 값이다).

8.53 1부터 3사이의 새값을 가지도록 하는 Response의 변이자 SetRed(), SetYellow(), SetGreen()를 수정하시오. 이 성원함수가 검사된다면 련습 8.52의 수정된 구축자가 필요한가?

8.54 RandomInt의 가상연산자를 다중정의하시오. 이 출력연산자를 그림으로 그려서 흐름의 값을 현시하시오.

제 9 장. 목 록

소개

많은 문제들에서 객체들의 배렬을 1차원목록 또는 다차원목록으로서 정의해야 할 필요가 제기된다. 이러한 과제들을 위해 C++는 2개의 표현을 제공한다. 하나는 배렬을 기초로 하고 다른 하나는 클라스를 기초로 한다. 배렬은 전통적으로 쓰인 표현이며 많은 서고들의 목록이 이 형태들로만 작업하기때문에 매우 중요하다. 그러나 C++는 이전의 C와 호환성을 보장하며 배렬에서 제한성을 극복하였다. 목록들의 클라스는 가장 잘 쓰이는 표현으로 되고 있다. 목록에 대한 여러개의 클라스표현들은 표준본보기서고(STL)에 정의되여 있다. 이 클라스들은 많은 값을 가질수 있기때문에 용기클라스(container class)라고 부른다. 가장 중요한 클라스는 vector클라스이다. 여기서는 목록에서 배렬과 vector객체들을 정의하고 처리하는 방법을 고찰한다.

주요개념

- 1차원배렬
- 배렬의 첨자화
- 파라메터로서의 배렬
- 파라메터로서의 배렬원소들
- 문자렬
- 표준본보기서고(STL)
- 용기클라스
- 본보기클라스
- 벡토르
- 벡토르첨자화
- 벡토르재편성
- 문자렬의 첨자화
- 반복자
- 반복자의 참조해제
- 벡토르들의 벡토르
- 정렬
- 함수 InsertionSort()
- 함수 QuickSort()
- 탐색
- 함수 BinarySearch()
- 알고리듬서고
- 함수 find()
- 표
- 행렬
- 성원초기화목록
- 다차원배렬

9.1 이름 붙은 모임

5개의 옹근수형객체 Value1, Value2,…, Value5에서 최소값을 구한다고 가정하자. 아래의 코드토막은 그 값을 계산한다.

```
int MinimumSoFar = Value1;
if(Value2< MinimumSoFar)
    MinimumSoFar = Value2
if(Value3< MinimumSoFar)
```

```
        MinimumSoFar = Value3
    if(Value4< Minimum SoFar)
        MinimumSoFar = Value4
    if(Value5< MinimumSoFar)
        MinimumSoFar = Value5
```

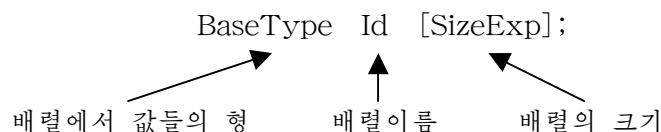
이 목록에서 객체이름은 비슷하지만 매 객체들이 다른 객체들에 대하여 전체적으로 독립이므로 여기서는 if문이 매 객체에 다 요구된다.

이제 1000개 객체들의 목록 Value1부터 Value1000까지중에서 최소값을 구한다고 가정하자. 객체들이 서로 독립이기때문에 4장에서 평균값문제에서처럼 반복구조를 도입할수 없다. 그리고 우와 같이 처리하면 매 변수에 대한 if명령문이 생기므로 코드서술이 길어 지고 오유가 생길수 있다. 이러한 결함을 없애기 위하여 같은 형의 변수들에 대해서 하나의 정의를 할수 있는 목록이 필요하다. 간단한 방법으로 목록의 원소들을 개별적으로 참조하는 기능이 필요하다.

목록형의 검사는 배렬을 가지고 시작하는데 배렬은 C++의 기본적인 목록기구이다. 그 검사후에 클라스 vector를 고찰하자. vector클라스는 vector객체들이 반복되는 원소들을 가질수 있기때문에 용기클라스라고 한다. vector클라스는 STL에서 정의된 여러개의 용기클라스들중의 하나이다. vector클라스를 리용하여 배렬과 련관된 일부 프로그람작성의 제한들을 압도할수 있다. 더우기 우리는 일부 공동프로그람작성에서의 오유를 피하는 벡토르기능을 사용할수 있다.

9.2 1차원배렬

배렬을 이루는 개별적인 객체들을 원소라고 한다. 배렬원소들은 집체적으로 또는 개별적으로 참조될수 있다. 그것들은 임의의 형, 지적자형, 앞서 정의한 파생형으로 될수 있다. 배렬에서 매 요소들은 다 같은 형이 되여야 한다.

다른 객체들처럼 배렬들은 초기화될수도 있고 초기화되지 않고 정의될수도 있다. 만일 초기화를 하지 않으면 1차원배렬정의는 다음과 같다.

괄호안에 배렬의 크기를 쓸수 있다. BaseType는 그 배렬에서 개별적인 원소들의 형(토대형:basetype)이며 Id는 그 배렬의 이름으로서 식별자이다. 그리고 SizeExp는 문자그대로 사용하거나 그 배렬에 있는 객체들의 번호를 렬거하기 위해 문자로부터 나온 상수를 리용하는 가능한 표현이다. 상수정의 실례를 아래에서 보자.

```
const int N=20;
const int M=40;
const int MaxStringSize =800;
const int MaxListSize =1000;
```

아래의 정의는 C++배렬을 정의한것이다.

```
int A[10];// 10개의 옹근수배렬
char B[MaxStringSize]; // 80개문자들의 배렬
float C[M*N]; // 800개의 실수들의 배렬
int Value[MaxListSize]; // 100개옹근수들의 배렬
```

A, B, C와 Value는 1차원배렬들이다. 이 객체들은 개별적인 배렬원소들의 모임이다. 매 배렬원소는 다른 객체처럼 쓸수 있는 하나의 객체이다. 개별적인 배렬원소들을 지정하는 동작을 첨자화(subscripting, indexing)라고 한다. 즉 괄호안의 수자는 배렬에 놓여 있는 원소들을 참조하는데 리용된다. 그리고 또한 배렬의 특별한 원소에 첨자를 주는데 괄호들이 리용된다. 배렬의 첫번째 원소는 첨자값 0, 두번째 원소는 첨자값 1 등을 가진다. 배렬 A에 대하여 마지막원소는 예약값 p를 가진다. 다음의 그림은 A의 표현을 보여 준다. A의 풀이표는 그 원소가 초기화되지 않은것을 가리킨다.

A	–	–	–	–	–	–	–	–	–	–
	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]

다음의 코드토막에서 A의 개별적인 배렬원소들에 대한 참조는 모두 정확하다.

```
int =7;
int j=2;
int k=4;
A[0]=1; //A의 원소 0에 값 1을 준다.
A[i]=5; //A의 원소 i에 값 5를 준다.
A[j]=A[i]+j //A의 원소 j에 A의 원소 i의 값에 3을 더한 값을 준다.
A[j+1]=A[i]+A[0] //A의 원소 j+1에 A의 원소 i의 값에 A의 원소 0의 값을 더하여 넣어 준다.
A[A[j]]=12; //A의 원소 A[j]에 값 12를 준다.
cout<<A[12]; //A의 원소 2를 현시한다.
cin>>A[k]; //A의 원소 k에 다음에 들어 오는 값을 준다.
```

만일 표준입력흐름으로부터 입력한 값이 3이라면 그때 전의 코드토막을 실행한 결과는 다음과 같다.

A	1	–	8	6	3	–	–	5	12	–
	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]

주의

좋지 못한 첨자화

C++는 적당한 배렬첨자가 사용된다는것을 담보하는 자동적인 방법을 제공하지 않는다. 실례로 다음의 코드토막은 오유통보가 발생하지 않는다.

```
int A[10];
int C[5];
B[5]=1;
```

오유통보대신에 우의 값주기의 적합한 결과는 대부분의 편집기들이 자기의 능동상태에서 배렬 A후에 즉시 B를 할당하기때문에 객체 A[5]의 기억위치가 수정되는것이다. 그러므로 B[5]는

A의 끝으로부터 15번째 원소를 참조하며 그때 A의 끝은 A［5］이다. 대부분의 공통적인 첨자오유는 배렬의 마지막원소를 잘못 참조하는것이다. 초학도나 혹은 경험 있는 프로그람작성자도 때로는 배렬의 마지막원소가 그 배렬의 크기보다 하나 작은 첨자를 가진다는것을 잊어 버리는 경우가 있다. 일부 적당한 첨자들의 리용을 담보하는 자동적인 방법이 없기때문에 쏘프트웨어개발자들은 표준본보기서고의 용기클라스들을 리용하기 시작하였다. 이 클라스들은 반복자들과 목록원소들에게 조종권호출을 제공하는 성원함수들을 제공한다. 첨자들이 자동적으로 검사되도록 이 클라스들을 쉽게 확장하는것은 가능하다.

9.2.1 배렬의 초기화

개별적인 배렬원소들은 다른 객체들과 비슷한 방식으로 초기화 된다. 특히 명시적인 초기화를 하지 않은 이상 전역적인 범위에서 기본형으로 정의된 배렬들의 매 요소들은 령으로 설정된다. 국부범위에서 기본형들로 정의된 배렬들을 초기화하지 않으면 배렬원소들은 초기화되지 않는다.

```
int Frequency [5]={0,0,0,0,0};
int Total[5]={0};
int Sum[5]({0,0,0,0,0});
int Count[5]({0});
```

frequency의 정의는 자기 원소모두를 령으로 설정한다. Total의 정의는 첫번째 배렬원소 Total[0]에 령을 설정하며 만일 부분적인 초기화만이 주어 지면 지정하지 않은 원소들은 령으로 설정된다. Sum과 Count의 정의는 초기화형태대신에 비슷하게 리용하고 있다. 다음의 배렬정의들로 명백하게 초기화를 한다. SizeExp가 제공되지 않기때문에 매개 정의에서 원소들의 변환초기화표현의 번호에 의하여 결정된다.

```
int Digits[]={0,1,2,3,4,5,6,7,8,9};
int Zero[]={0};
char Alphabet[]={'a','b','c','d','e','f','g','h','i',
  'j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z'};
```

배렬 Digits의 정의는 Digits［0］에 0을, Digits［1］을 설정하는 방법으로 10개 원소배렬을 초기화한다. Zero의 정의는 Zero[0]을 0으로 초기화하여 1개의 원소배렬을 초기화한다. Alphabet정의는 Alphabet[0]은 'a'로, Alphabet[1]은 'b' 등으로 초기화하여 26개 원소의 배렬을 정의한다.

배렬이 클라스기본형으로 정의될 때 초기화는 언제나 그 범위안에서 진행된다. 그 클라스에 대한 기정구축자는 자동적으로 매개의 개별적인 배렬원소에 적용된다. 기정구축자가 적용되기때문에 명백한 초기화없이 주어 진다. 아래의 실례는 10개의 Rational객체들의 배렬 R를 정의한다. 매개의 묶음원소는 0/1의 암시적인 Rational값으로 초기화된다.

```
Rational［10］;//10개의 Rational들은 0/1로 초기화된다.
```

9.2.2 상수배렬

다른 객체정의들에서처럼 변이자 const는 하나의 배렬정의안에서 적용될수 있다. 변이자는 보통 초기화한후에 효과를 가지는데 그 객체는 하나의 상수처럼 취급된다. 이 경우에 개개의 배렬원소들의 값들은 변경할수 있다. 다음의 실례는 2개의 원소배렬 B를 정의하고 초기화한다.

```
const int B[2]={10,100};
```

이렇게 되면 배렬원소의 어느것도 변경될수 없다.

```
B[0]=20;    // 오유;상수객체는 값을 할당할수 없다.
cin >> B[];    // 오유;상수객체는 지령행에서 값을 받을수 없다.
```

const배렬원소는 하나의 값이 요구될 때만 리용된다.

```
const int i=B[1];    // 옳음:i는 B[1]의 복사이다.
cout<<B[0];    // 옳음:B[0]은 표준출력흐름으로 나간다.
```

9.2.3 간단한 배렬처리

아래의 코드토막은 표준입력으로부터 MaxListsize값을 얻고 Value배렬에 그값을 대입한다.

```
const int MaxListSize =10;
int Value[MaxListSize];
int n=0;
int CurrentInput ;
while ((n<MaxListSize)&&(cin>>CurrentInput))
{
    Value[n]=CurrentInput;
    ++n;
}
```

배렬 Value는 MaxListSize개만한 요소를 가진다. 리용자가 여러개 값들을 제공할 필요가 없을 때에 객체 n은 값들이 할당된 목록에서 원소번호들을 표현하는것으로 정의한다. Value[0]부터 Value[n-1]로 처리가 진행된다. 새로 입력된 값을 기억하는 장소는 Value[n]이다. 코드토막의 시작에서 할당되지 않은 n번째 요소는 0으로 초기화된다. Value[n]은 CurrentInput값으로 초기화된다.

while명령문은 입력값을 처리하는데 리용된다. 만일 조건식에서 2개의 항목이 다 성립하면 while순환이 반복된다.

항목 (n<ListSize)가 참일 때는 값을 저장할 Values에 할당해제된 원소들이 있다는것을 지적한다. 론리적인 표현으로서의 배렬순서평가로부터 만일 이 항목이 거짓이면 두번째 항목은 평가하지 않는다.

항목 (cin>>CurrentInput)은 참일 동안 들어 오는 값이 있다는것을 표시한다.

while순환이 실행되면 다음의 배렬원소는 현재입력값을 보관하는데 쓰인다.

```
Values[N]=CurrentInput ;
```

이렇게 할당된 다음 n은 또 다른 값을 목록에 값을 추가하기 위하여 증가된다.

```
++n;
```

while검사식은 그때 재평가되여 그것이 참이면 그 처리는 반복된다. 실례로 입력흐름은 다음과 같다고 하자.

```
4  9  5
```

처음에 순환부를 거치면 n은 령이며 따라서 Values[0]에는 입력값 4가 할당된다. 현재 배렬 Values의 기억기는 아래와 같이 된다.

Values	4	-	-	-	-	-	-	-	-	-

다음 반복하여 n은 1로 증가될것이며 Values [1] 에 입력값 9가 할당된다.

Values	4	9	-	-	-	-	-	-	-	-

계속 반복하여 n은 2로 증가될것이며 Values[2]는 5로 된다.

Values	4	9	5	-	-	-	-	-	-	-

배렬원소에 대한 입력값의 매개의 값주기후에 n을 증가하면 n은 설정된 values에서 원소들의 번호로 된다.

만일 n이 증가된후에 MaxListSize와 같다면 검사식은 거짓이며 순환은 끝난다. 추가한 입력값을 보관하는 Values에서 할당되지 않는 원소가 더는 없기때문에 끝낼것을 요구한다. 만일 입력흐름이 초기에 비였다면 순환은 진행되지 않으며 이 경우 정확한 값을 귀환하여 0으로 남아 있게 된다. 이와 같이 순환이 어떻게 몇번 실행되는가 하는것은 문제가 아니며 순환이 끝나면 n은 목록에 할당된 값의 번호를 돌린다.

다음의 실례는 n개의 원소들을 가진 배렬 List에서 지정한 값을 탐색한다. 탐색되는 값을 일반적으로 열쇠(Key)라고 부른다.

```
cout << " Enter the search value (number):" ;
int Key ;
cin>>Key ;
int i;
for(i=0;<i<m&&(List[i]!=Key);++i)}
    continue;
}
if(i==m)
    cout<<Key<<" is not in the list" <<end1;
else
    cout<<Key<<" is" << i  <<" th element"<<end1;
```

m은 설정된 List에서 배렬원소들의 수이다. 배렬원소들 List[0]…List[m-1]만이 검사되여야 한다. 설정된 배렬원소의 개수가 m이라는것을 알고 있기때문에 for명령문을 하나씩 증가하여 0부터 m-1까지의 배렬원소들의 상태를 쉽게 표현할수 있다. 첨자값을 표현하기 위해 첨자 i를 리용한다. i의 값은 순환한 다음 유효하여야 하기때문에 for순환고리안에서 그것을 선언하면 안된다.

for순환검사식은 현재배렬원소 List [i] 를 처리한다. Value로 초기화된 while순환에서처럼 두 항목들이 반복되는 순환에 대하여 참이여야 한다. for순환에 대한 항목들은 아래와 같다.

항목(i<m)이 참일 동안은 열쇠값과 비교하든 하지 않든 검사를 위한 List에서의 현재원소라는것을 표시한다. 만일 이 항목이 거짓이면 두번째 항목은 평가되지 않는다.

항목(List[i]!=Key)이 참일동안 현재원소가 열쇠값과 같지 않다는것을 표시한다. 만일 검사식이 참이면 List의 다음 원소값에서 열쇠값에 대한 탐색을 계속한다. 검사식이 다음평가를 설정한 for순환본체자체에서는 i를 증가할 필요가 없다. 순환고리가 어떤 동작도 하지 않게 하는 방법에는 여러가지가 있다. 실례로 C++에서 빈 명령은 합법적이므로 그 순환은 다음과 같이 표현될수 있다.

```
for(i=0 ; (i < M) && (List[i] != Key) ; ++i) {
    continue;
}
```

continue명령에서 예약어 continue를 사용하여 진행한다.

```
for (i=0 ; (i < M) && List) != Key) ; ++i}
    continue;
}
```

순환에서 continue명령문은 그 본체의 실행이 이 반복자에 의하여 끝나게 한다. 순환고리에서 continue명령문이 실행될 때 조종은 순환고리의 마지막에 넘겨 지게 된다. 앞서 서술한 for순환에서 i의 값이 M보다 작을 때까지 순환이 계속되며 i가 M과 갈다면 조종은 다음상태로 넘어 간다.

```
++i
```

for순환이 끝난 다음에는 i의 값을 검사하여 Key와 같은 값을 가지는 List배렬원소들중의 하나를 결정할수 있다. 만일 i가 값 m으로 된다면 모든 배렬원소들이 검사된것으로 되며 순환은 끝난다(List[0]~List[m-1]이 참조된 원소들이라는것을 기억하시오). 이 경우 열쇠값이 나타나지 않으며 적당한 통보가 출력흐름으로 나가게 된다.

i가 m의 값을 가지지 않는다면 (List[I]!=Key)는 0부터 -1사이의 값을 가지는 어떤 i에 대하여 거짓이기때문에 for순환은 끝나는것으로 되여야 한다. 이 경우 List[i]는 열쇠값이며 적당한 통보는 표준출력흐름으로 나가게 된다.

현재 List, m, Key 그리고 i에 대하여 다음의 표현을 리용하여 그 코드토막을 거쳐 확인한다. 실행항목에서 for순환을 시작하려고 한다.

List	4	9	5	-	-	-	-	-	-	-
m	3									
Key	5									
i	-									

for순환의 초기화단계는 처음에 실행되고 첨자 i의 값은 0으로 할당된다.

List	4	9	5	-	-	-	-	-	-	-
m	3									
Key	5									
i	0									

이때 검사식이 평가된다. List의 m, Key, i의 값에 기초하여 두 항목들은 다 참이다(0은 3보다 작고 4는 5와 같지 않다). 첨자 i는 이때 다음평가준비를 위하여 1로 증가된다.

List	4	9	5	-	-	-	-	-	-	-
m	3									
Key	5									
i	1									

그다음 검사식이 재평가된다. List, m, Key, i의 값에 기초하여 두 항목은 다시 참으로 된다(1은 3보다 작고 9는 5와 같지 않다). 첨자 i는 다음평가준비를 위하여 2로 증가된다.

List	4	9	5	-	-	-	-	-	-	-
m	3									
Key	5									
i	2									

검사식은 다시 평가된다. List,m,Key,i의 값에 기초하여 첫번째 항목은 참으로 되지만 두번째 항목은 현재 거짓으로 된다(2는 3보다 작지만 5와 5는 같다). 그 항목들이 둘다 참이여야 순환을 반복하기때문에 **for**순환은 끝나게 된다.

if명령이 그때 평가되고 **if**문의 검사식이 거짓으로 된다(2는 3과 같지 않다). 결국 아래와 같은 출력을 발생한다.

```
5 is 2th element
```

코드토막을 탐색하는 열쇠값은 배렬처리에서 대표적이다. 배렬처리준비를 위한 초기화, 차례로 배렬원소들을 처리하는 순환, 목록의 처리가 완성되였는가,안되였는가를 알기 위한 검사가 있다. 실례로 아래의 코드토막은 목록크기 n이 1인 목록 Values의 최소값을 찾는다.

```
int MinimumSoFar = Values[0];
for ( int i = 1; i < n; ++i)
    if (Values[i] < MinimumSoFar) {
        MinimumSoFar = Values[i];
    }
}
```

배렬에서 최소값을 찾자면 차례로 매 원소들을 검사하는것이 필요하다. 만일 코드토막이 지금까지 본 최소배렬원소값을 계속 알고 있다면 그 값이 보다 더 작은 최소값이다. 처리가 매 배렬원소에 대하여 진행된다면 마지막배렬원소를 참조한 다음에는 이제까지 본 최소값이 실제최소값이다. 임의의 크기를 가진 목록의 최소값을 구하는 우의 코드토막은 이장의 앞에서 서술한 5개의 값의 최소값을 구하는 코드토막보다 더 작다는데 주의를 돌리시오.

아래의 코드토막은 앞의 코드토막에 대한 해설을 주었다. 번호를 단 해설들은 앞의 론의를 반영한다.

```
int MinimumSoFar  = Values[0];
// (1):
// Values[0]
```

```
for (int i = 1; i < m; ++i) {
    // (2):
    // Values[i-1]
    if (Values[i] < MinimumSoFar) {
        MinimumSoFar = Values[i];
    }
    // (3):
    // Values[i]
}
// (4):
// Values[m-1]
```

두번째 해설은 그 점에서 i-1이 령이고 첫번째 해설이 참이기때문에 for본체의 처음 실행값은 참이다. 사실 만일 List[i]가 Value[0]~Vlaue[i]중에서 최소값이면 세번째 해설은 MinimumSoFar를 갱신한 if문때문에 두번째 해설이 참이다. 만일 for본체가 반복되면 그 점에서 파일이 증가된 i의 값을 리용하여 이전 반복자로부터 3번째 해설까지의 재설명이 쉽다. 4번째 해설은 세번째 해설의 재설명(만일 m이 1보다 더 크다면)이든가 혹은 첫번째 해설의 재설명(만일 m이 1이라면)이 단순하기때문에 진실이다. 4번째 해설이 참이므로 그 목록에서 최소값을 정확히 결정하였다.

9.2.4 문자렬배렬

C++는 문자렬들을 표현한 char형배렬의 초기화형식을 제공한다. 실례로 아래의 Letters정의는 27개 원소배렬을 정의하고 초기화한다.

```
char Letters[] = "abcdefghijklmnopqrstuvwxyz";
```

그 배렬의 첫 26개 원소를 즉 Letter[0]-Letter[25]는 초기화렬에서 대응하는 위치로부터 자기의 초기화문자를 얻는다. 즉 배렬원소 Letter[0]은 문자렬에서 첫 문자 'a'를, 두번째 배렬원소 Letter[1]은 문자렬에서 'b'를, …26번째원소 Letter[25]는 문자렬의 26번째 문자 'z'를 얻는다.

마지막배렬원소 Letter[26]는 빈 문자 '0'으로 초기화된다. 배렬원소 Letter[26]은 C++언어가 자동적으로 이 값을 제공한다. 다음의 배렬정의는 Greetings[0]은 'H'로 초기화되고 Greetings[1]은 'e'로,…, Greetings[11]는 'd'로, Greetings[12]은 '0'로 초기화된 13개의 원소배렬을 창조한다.

```
char Greetings[ ]="Hello, world";
```

입출력흐름서고는 char형배렬인 오른쪽연산수에 대한 입력연산자 <<의 정의를 포함한다. 그 조작은 배렬의 첫 null문자의 앞에 있는 모든 문자를 표시한다. 실례로 다음의 삽입은 표준출력흐름에서 표시되는 단어 "Hello, world"를 나타낸다.

```
cout << Greetings;
```

추출연산자는 하나의 char형배렬인 오른쪽연산수에 대한 입출력흐름서고에서 정의되기도 한다. 암시적인 추출은 처음에 빈 공백문자앞에서 뛰여 넘고 다음 입력흐름으로부터 비지 않은 공백문자를 련이어 추출한다. 효력이 없는 '\0'은 마지막에 추출된 문자의 다음에 자동적으로 보관된다. 만일 표준입력흐름이

how are you today? I am fine

을 포함한다면 아래의 코드토막은 해당 공백 없는 문자렬 하나를 현시할것이다.

```
const int MaxStringSize=10;
char s[MaxStringSize];
while(cin >> s)    {
    cout << s << end1;
}
```

how
are
you
today?
I
am
fine

S와 련관된 기억기는 아래와 같다.

S	'f'	'f'	'n'	'e'	'\0'	'?'	'\0'	-	-	-

물음표기호와 두 문자렬에서 2개의 끝문자는 전체 추출이 4개의 문자만 포함되였기때문에 s에 배렬값으로서 남아 있다. 비록 추출연산자가 char형배렬객체를 가지고 작업하도록 정의되였다고 하여도 그 사용은 일반적으로 무효로 된다. 대신에 문자들은 문자로서 명백하게 추출된 문자이고 배렬에 대한 할당이든가 입력흐름성원함수 get() 또는 getline()의 높은 다중정의방식의 하나가 리용되는 어느쪽이든 하나이다.

성원함수 get()의 방식은 2개의 필요한 파라메터와 하나의 선택파라메터를 가진다(다른 입출력흐름성원함수의 표현에 대하여서는 부록 2를 보시오). 필요한 파라메터들은 char형배렬과 같이 있다. 선택파라메터는 기정값이 새로운 행 '\n'인 char형이다. 이 기호를 구분기호(delimiter)라고 한다. 문자들은 표준입출력흐름으로부터 추출되며 다음의 조건을 만족할 때까지 다음의 유효한 배렬원소에 할당된다.

- 추출하려는 문자가 더는 없다. 즉 파일끝이다.
- 추출되는 다음문자는 경계로 된다.
- 추출된 문자들의 개수는 두번째 파라메터의 값보다 한 문자가 작다.

이 세가지 조건이 문자렬추출처리동작을 완료한다면 빈 문자 '\0'은 마지막으로 추출된 값을 나타내는 다음의 배렬요소에 보관된다. 입력흐름성원함수 getline()은 구분기호를 만나면 추출되는것을 제외하고 앞서 서술한 성원함수 get()와 비슷한 방식으로 조작한다. 그러나 구분기호는 그것의 배렬파라메터에 할당되지 않는다.

만일 표준입력흐름에

A

```
multi-line
 example.
```

이 포함되면 코드토막

```
const int MaxStringSize=10;
char S[MaxStringSize];
while (cin.getline(s,MaxStringSize))}
    cout<<s<<end1;
}
```

은

```
A
multi-lin
e
 example.
```

을 생성한다. 작성후에 배렬 S와 련관된 기억기는 다음과 같이 된다.

S	' '	'e'	'x'	'a'	'm'	'p'	'l'	'e'	'.'	'\0'

경험

문자렬을 쓰겠는가 아니면 string클라스를 쓰겠는가?

이 장의 앞에서 문자렬처리가 요구되였을 때 문자렬서고로부터 string클라스를 사용하였다. 다음장들에서도 우리는 string클라스가 더 확장된 기능모임을 가지기때문에 계속 그것을 리용하기로 한다. 문자렬들에 기초한 유산서고함수들의 호출이 요구될 때만 문자렬표현을 사용한다.

지금까지의 실례에서는 국부변수들로서 배렬을 사용하여 왔다. 다음부분에서 파라메터들로서 배렬을 가진 여러개의 쓸모 있는 함수들을 개발한다. 이 함수들은 목록추출, 목록삽입, 목록탐색을 수행한다.

문제

1. 배렬로 이루어 진 개별적인 객체들을 무엇이라고 하는가?
2. 2000개의 옹근수들을 가지고 Scores라는 이름을 가진 배렬을 정의하기 위한 적당한 선언을 주시오.
3. 개별적인 배렬원소들을 호출하는데 사용된 조작의 이름은 무엇인가?
4. StackObject형의 50개 객체를 가질수 있는 StackElements라는 배렬을 정의하기 위한 적당한 명령을 쓰시오.
5. 500개의 단정확도류동소수점객체를 가질수 있는 GradePointAvg라는 배렬을 정의하기 위한 적당한 연산을 쓰시오.
6. 아래의 정의를 보시오.

```
const int MaxSize = 200;
int Hits[MaxSize];
int i, j, k;
```

Hits의 원소 k에 6을 더하는 C++명령을 쓰시오.
원소 k+1에 Hits의 원소 i를 복사하는 C++명령을 쓰시오.
Hits의 i와 k원소를 합하고 그 결과를 Hits의 j+5원소에 넣는 C++명령을 쓰시오.

7. 아래의 배렬정의는 전역범위에서 주어 졌다고 가정하자.

```
const int MaxSize=100;
int Points[MaxSize];
```

Point원소들의 초기값들은 무엇인가?

8. 아래의 배렬정의는 국부범위에서 주어 졌다고 가정하자.

```
const int MaxSize=30;
float Timestamp[MaxSize];
```

Timestamp의 원소들의 초기값들은 무엇인가?

9. 10개 원소들을 가진 Distance라는 배렬에 대한 배렬정의를 하시오. Distance의 매개 원소는 그 배렬에서 그의 위치의 값으로 초기화되여야 한다. 실례로 Distance [6] 은 6으로 초기화되여야 한다.
10. 전역범위에서 아래의 정의를 만든다고 생각하자.

```
const int MaxSize=6;
int Weights[MaxSize]=[3,4];
```

배렬 Weights의 초기값을 주시오.

11. 아래의 정의를 보자.

```
int Buttons[]={5,6,10,12,13};
```

몇개의 원소들이 Buttons배렬에 포함되는가?

12. 다음의 값 23.1, 34.5, 35.6, 28.0, 38.2, 91.3으로 초기화된 Coefficients배렬에 대한 물음정의를 쓰시오.
13. 문자렬 "Madam Bovary"로 문자배렬 Title을 초기화하는 배렬정의를 쓰시오.

9.3 파라메터로서의 배렬

C의 초기개발자들은 효과성에만 관심을 두었다. 이런 리유로 하여 그들은 배렬형을 제1클라스형(first-class type)으로 만들지 못하였다. 다시 말하여 배렬객체는 다른 형의 객체들에 적용될수 있는 언어기능으로 리용되지 못하였다. 이러한 제한성은 다음과 같은 3가지 방식에서 C++언어에 그대로 넘겨졌다.

- 함수귀환형은 배렬으로 될수 없다.
- 배렬파라메터는 참조파라메터로만 될수 있다.
- 배렬은 값주기표식으로 될수 없다.

이 제한성들을 극복하는것은 개별적인 배렬원소들의 복사를 할것을 요구하였다. 그 요구는 배렬이 많은 원소들로 구성된다면 기억기와 처리시간에 아주 품이 많이 들게 될수 있다. 배렬파라메터에 대한 정의는 참조로만 전달되게 될수 있는 효과가 있다. &는 배렬파라메터의 정의에서 요구되지 않거나 혹은

접수되지 않는다.

배렬파라메터정의에 대한 정의는 괄호안에서부터 나올수 있다. 1차원배렬을 허락한다. 이 신축성은 같은 함수가 여러가지 크기의 배렬들을 처리할수 있다는것을 의미한다. 비록 함수들이 파라메터선언부만한 배렬크기를 요구하지 않지만 다른 파라메터는 일반적으로 얼마나 많은 배렬의 원소들을 처리할것인가를 가리킨다. 3개의 형식파라메터를 가진 void함수 GetList()를 정의한다. 이 함수는 입력흐름 cin으로부터 값들을 추출하며 배렬에 그 값들을 보관한다.

```
void GetList(int A[],int MaxN,int &n);
    for(n < MaxN) && (cin >> A[n]); ++n) {
        continue;
    }
}
```

첫 파라메터는 추출된 값을 가지는 배렬 A이다. 두번째 파라메터 MaxN은 추출되는 값들의 최대번호이다. 세번째 파라메터 n은 실행되여 추출된 값의 번호를 가리킨다. 만일 cin으로부터 충분한 입력값들을 입력하지 않는다면 MaxN은 N과 차이나게 된다. A와 n은 둘 다 참조파라메터들이다. 그것은 A는 배렬이 항상 참조되기때문이며 n은 참조파라메터구성이기때문이다.

함수 getlist의 본체는 하나의 for문으로 이루어 진다. 열쇠탐색코드토막에서처럼 동작이 for순환본체안에서 요구되지 않는다. 순환검사식은 반복되는 순환에 대하여 참이여야 하는 2개의 항을 가진다. 검사식은 short-circuit평가사용을 위해 설계되였다.

첫항은 지금까지 추출된 값들의 번호가 추출하려는 값들의 최대번호보다 더 작아야 한다는 조건이다. 다만 이 항목이 참이면 두번째 항이 검사된다. 두번째 항의 검사는 A의 리용가능한 다음원소에 1개 값을 할당하는 cin으로부터 시도되는 추출을 일으키는데 그것은 A[n]이다.

만일 그 추출이 성공적이지 못하면 연산값은 0이면 false이며 순환의 끝시점에서 결과를 낸다. 만일 추출이 성공이면 연산값은 령이 아니며 그 값은 참이다. 이렇게 값이 원소 A[n]에 할당되며 전체 검사식은 참이다. 만일 검사식이 참이면 n은 증가된다. 증가된다는것은 또 다른 값이 배렬에서 추출되고 할당되였다는것을 말한다. 표준입력흐름이 다음의 값들을 입력한다고 가정하자.

```
6 9 82 11 29 85
11 28 91
```

이때 코드토막

```
const int MaxListSize=10;
int Scores [MaxListSize];
int NbrScores;
Getlist(Scores,MaxListSize,NbrScores);
```

은 다음의 방식대로 배렬 Scores를 할당한다.

Scores	6	9	82	11	29	85	11	28	91	-

Scores는 형식참조파라메터배렬 A의 원소를 설정할 때 대응하는 Scores의 원소를 실제로 변환한다.

또한 앞의 코드토막은 NbrScores를 9로 설정한다. GetList의 호출은 실지배렬파라메터 Scores넘기기에서 괄호를 사용하지 않는다는것을 주의하여야 한다. 조종코드토막은 Scores가 배렬이라는것을 안다. 그러므로 괄호는 필요하지 않다. 고찰하려는 다음함수는 PutList()이다.

```
void PutList(const  int A[], int n) {
    for(int i=0 ; i < n; i++):
        cout<<A[i]<<end1;
    }
}
```

이 void함수는 2개의 형식파라메터를 가진다. 첫번째 파라메터는 현시하려는 int형배렬이다. 목록을 현시하는것은 그 원소들에 대한 그 어떤 변경을 요구하지 않기때문에 수식 const가 리용된다. 두번째 파라메터는 표시하려는 배렬에서 원소들의 수를 표현하는 파라메터이다.

이 함수는 표준흐름에 행 하나당 n개 배렬원소값을 반복 표시하기 위하여 for문을 사용한다. 실례로 아래의 호출은 앞에서 정의된 Scores표배렬을 말한다.

```
PutList(Scores, NbrScores);
```

이 호출은 그의 출력수만큼 발생한다.

```
6
9
82
11
29
85
11
28
91
```

다음에 int형함수 Search()를 고찰하는데 Search()함수는 3개의 가상파라메터를 가진다. 첫번째 파라메터는 int형배렬 List이고 두번째 파라메터는 고찰되는 원소수 n이며 세번째 파라메터는 열쇠의 값 Key이다.

```
int Search(const int List[],int m,int Key) {
    for (int i=0; i < m; ++i)}
        if(List[i] == Key)  {
            return i;
        }
    }
    return m;
}
```

함수 Search()는 9.2.3부분에서 열쇠값에 대하여 배렬을 탐색한 코드토막과 비슷하다. 그러나 함수 Search()는 값을 찾을 때마다 표시하는 통보를 현시하지 않는다. 만일 Search()가 목록에서 값을 찾는

다면 함수는 첫번째로 비교된 원소의 초기값을 귀환한다. 열쇠값이 배렬원소들속에 있다면 함수는 목록에서 원소들의 수 m을 돌려 준다. 목록원소들이 배렬에서 0~m-1의 초기위치를 차지하기때문에 값 m은 열쇠값이 그 항목에 없다는것을 가리킨다. 다음의 코드토막에서는 앞서 정의된 Scores배렬이 2개의 값에 의해서 탐색 되였다.

```
int i1= Search(Scores,NbrScores,11);
int i2= Search(Scores,NbrScores,30);
```

Scores배렬에서 값 11의 첫번째 비교를 Scores[3]을 가지고 하기때문에 첫번째 호출은 i1을 3으로 초기화한다.두번째 호출은 Scores[0]부터 Scores[8]까지의 배렬이 값 30을 가지지 않기때문에 i2를 9로 초기화한다.

비록 배렬이 실제파라메터로서 리용될 때 첨자들이 사용되지 않아도 그것들은 개별적인 배렬원소들이 실제파라메터로 넘겨 질 때 사용된다. 첨자로써 개별적인 원소를 지정하므로 여기서는 괄호를 리용해야 한다. 실례로 프로그람 9-1의 함수 main()은 6장에서 정의된 Swap()함수를 호출한다. Swap()함수는 int형배렬 Number로부터 원소들의 한쌍의 값을 서로 교환한다.

```
Swap(Number[i], Number[n-i]);
```

Swap()함수정의는 2개의 인수가 int형참조파라메터들이라는것을 기록한다. Number의 기본형이 int형이기때문에 Swap()함수에 대한 2개의 Number배렬원소를 넘기는것이 적당하다. 프로그람 9-1은 함수 GetList()를 리용하여 배렬 Number를 초기화한다. 목록에서 값들의 순서를 바꾼후에 PutList()는 목록을 표시하는데 리용된다.

```
// Program9.1:
#include<iostream>
#include<string>
using namespace std;
void GetList(int A[], int MaxN, int &n);
void Swap(int &Value1, int &Value2);
void PutList(const int A[], int n);
int main() {
    const int MaxListSize = 100;
    int Number[MaxListSize];
    int n;
    GetList(Number, MaxListSize, n);
    for(int i = 0; i < n/2; ++i) {
        Swap(Number[i], Number[n-i]);
    }
    PutList(Number, n);
    return 0;
}
void GetList(int A[], int MaxN, int &n) {
```

```
    for ( n = 0; (n < MaxN) && ( cin >> A[n]; ++n) {
    continue;
    }
}
void Swap(int &Value1, int &Value2) {
    int RememberValue1 = Value1;
    Value1 = Value2;
    Value2 = RememberValue1;
}
void PutList(const int A[], int n) {
    for(int i = 0; i < n; ++i) {
    cout << A[i] << endl;
    }
}
```

프로그람 9-1. 입력한 값을 반대순서로 표시하기

9.4 정 렬

4장에서는 프로그람 4-3에서 정렬에 대한 일반개념을 소개하였는데 이 프로그람은 증가하는 순서로 3개의 입력값들을 표시하였다. 복사값들을 얻기 위해 증가하지 않고 비감소항을 사용하였다. 여기서 비감소순서로 임의의 크기를 가진 목록에서 값들을 묶는 일반적인 정렬문제를 보기로 하자.

정렬은 흔히 매개의 순환우에서 목록의 일부 값들을 재배렬하는 반복처리이다. 실례로 순환 i에서 SelectionSort()로 소개된 방법은 A의 제일 작은 요소를 찾고 그 요소의 값을 A[i]요소의 값과 교체한다. 또 다른 실례에서는 순환 i에서 InsertionSort()로 알려 진 방법은 요소 A[0]~A[i-1]에 기억된 값들에 대하여 A[i]의 값을 정확하게 배치한다.

일부 정렬들은 오히려 반복적이기보다 재귀적이다. 재귀적방법은 일반적으로 목록을 개별적으로 정렬되는 부분목록(sublist)들로 분해한다. 만일 그 과제를 완성하는것이 필요하다면 정렬된 부분목록들은 정렬된 목록안에 섞이게 된다. 여기서는 반복적인 정렬 InsertionSort()를 고찰한다.

이 장의 마지막에서 재귀적정렬QuickSort() 를 고찰한다. 함수 SelectionSort()와 또 다른 정렬은 련습에서 고찰하게 된다.

아래의 론의들에서는 정렬하려는 목록을 char형값들의 목록이라고 가정한다. 같은 정렬들은 값의 다른 형들에 대하여 쉽게 수정될수 있다.

9.4.1 InsertionSort()방법

반복 i에서 InsertionSort()의 과제는 이전에 묶어 진 A[0]~A[i]의 요소값들에 대하여 정확하게 A[i]요소값을 배치하는것이다. 실례로 정렬하려는 목록이

A	'Q'	'W'	'E'	'R'	'T'	'Y'	'U'	'I'	'O'	'P'

이고 그것들속에서 정확하게 첫 7개값들을 배치한 순환이 완성되였다고 가정하자. 그 시점에서 목록은 아래와 같이 된다.

A	'E'	'R'	'Q'	'T'	'U'	'W'	'Y'	'I'	'O'	'P'

i가 7인 다음반복에서는 A[0]부터 A[6]의 정렬된 값들에 관하여 A[7]에 정확히 값을 배치하여야 한다. 이 경우에 A[7]의 값 'I'는 A[0]의 값 'E'와 A[1]의 값 'R'사이에 놓여야 한다. 그렇게 하기 위하여 먼저 림시객체 V에 A[7]의 값을 먼저 복사하고 그다음 A[1]부터 A[6]의 값들을 밀기한다. 그 시점에서 목록은 아래와 같이 된다.

A	'E'	'R'	'R'	'Q'	'T'	'U'	'W'	'Y'	'O'	'P'

요소 A[1]과 A[2]은 둘다 값 'R'를 포함한다는것을 관찰해 보시오. 이것은 A[i],A[o]의 요소가 오른쪽으로 한 위치만큼 밀기되였기때문이다. 순환을 완성하기 위해서 A[1]에 객체 V로부터 값 'I'를 복사한다.

A	'E'	'I'	'R'	'Q'	'T'	'U'	'W'	'Y'	'O'	'P'

비록 밀기처리를 수행하는 여러가지 방법이 있다 해도 제기된 방법은 A[i]와 A[i-1]을 비교하여 시작한다. 만일 이 두 요소들이 정확한 순서로 있다면 순환 i에 대해 밀기가 필요없다. 그것은 목록의 i개 요소들이 처음에 이에 정렬된 순서로 놓여 있기때문이다.

즉 처음 i-1개 요소들로 구성된 부분목록은 이미 정렬된 순서로 되여 있으며 A[i]의 값은 그 부분목록에서 제일 큰 값보다 더 크지 않다. 만일 A[i]가 A[i-1] 보다 더 작지 않으면 그때 A[i]안에서 배치되는 정확한 값에 대한 칸을 만들기 위해 A[i]의 복사칸 V를 만든다. 어느 요소값들이 밀기를 요구하는가를 결정하는것은 간단하다. 색인변수 i는 A에서 요소가 다른 밀기의 목표로 될것인가를 돕지 않는데 리용된다.

j의 초기값은 i이고 밀기하려는 첫번째 값은 A [j-1] 이다. 밀기가 일단 수행되면 j는 감소된다. j가 감소되는것은 그의 새로운 값이 밀기목표로서 현재 리용가능한 A에서 위치의 색인이거나 만일 적당하다면 값 V의 위치로서 현재 리용가능한 A에서 위치의 색인이기때문이다.

A검사는 어느 경우가 적용되는가를 결정하기 위해 진행한다. 만일 A[j-1]이 존재하고 V가 A[j-1]보다 작다면 밀기와 비교처리를 반복한다. 다른 방법으로 값 V에 대한 오른쪽위치가 알려 졌으면 밀기처리는 끝나게 된다.

실례로 i가 현재 8이고 A[j]가 있는데 A[j]는 'O'이고 A[j-1]보다 작으면 A[j-1]은 'V'이다. A밀기처리는 진행되여야 한다. 'O'의 A복사는 진행되고 V에 배치되며 첨자 j는 8로 결정된다.

V	'0'									
A	'E'	'I'	'R'	'Q'	'T'	'U'	'W'	'Y'	'O'	'P'

A[j-1]의 밀기와 j의 감소는 다음상태에서 결과를 낸다.

V | '0' |
|---|

A

'E'	'I'	'R'	'Q'	'T'	'U'	'W'	'Y'	'Y'	'P'

'O'가 'W'보다 작기때문에 또 다른 밀기가 진행되고 A[j]에 W를 복사한다. 마지막에 j가 감소된다.

V | '0' |
|---|

A

'E'	'I'	'R'	'Q'	'T'	'U'	'W'	'W'	'Y'	'P'

또 'O'는 'U'보다 작기때문에 또 다른 밀기를 진행하며 A[j]에 'U'를 복사한다.

V | '0' |
|---|

A

'E'	'I'	'R'	'Q'	'T'	'U'	'U'	'W'	'Y'	'P'

'O'는 'T'보다 역시 작기때문에 밀기를 진행하고 A[j]에 'T'를 복사하고 첨자 j는 감소된다.

V | '0' |
|---|

A

'E'	'I'	'R'	'Q'	'T'	'T'	'U'	'W'	'Y'	'P'

'O'는 ' Q'보다 작기때문에 밀기를 진행하며 A[j]에 'Q'를 복사하고 첨자 j는 다시 감소된다.

V | '0' |
|---|

A

'E'	'I'	'R'	'Q'	'Q'	'T'	'U'	'W'	'Y'	'P'

'O'는 역시 'R'보다 작기때문에 밀기를 진행하고 A[j]에 'R'를 복사한 다음 첨자 j는 다시 감소된다.

v | '0' |
|---|

A

'E'	'I'	'R'	'R'	'Q'	'T'	'U'	'W'	'Y'	'P'

'O'는 'I'보다 작지 않으므로 밀기가 필요하지 않으며 대신에 '0'는 V로부터 A[j]로 복사된다.

v | '0' |
|---|

A

'E'	'I'	'O'	'R'	'Q'	'T'	'U'	'W'	'Y'	'P'

목록 9-1은 앞서 론의한 void형함수 InsertionSort()를 정의한다. for순환고리부에서 바깥순환은 i에 의하여 여러가지 값들을 발생하고 내부순환 do명령문은 고리부안에서 밀기처리를 진행한다.

론리식에 들어 있는 두 항목들은 do순환고리가 다시 반복되든 반복하지 않든 조종한다. 첫번째 항목은 j가 령보다 더 큰가 아닌가를 결정한다. 만일 이 항목이 참이면 요소 A[j-1] 은 비교에 대하여 유용하다. 즉 만일 j가 령이라면 밀기는 끝나는데 요소 A[j]는 A[1]이며 A[j]는 V에게 적합한 위치이다.

두번째 항목은 A[j-1]이 V보다 더 큰가 작은가를 결정한다. 이 항목이 참이면 추가적인 밀기가 필요하다. 만일 그것대신에 A[j-1]이 V보다 더 크지 않다면 더이상 밀기는 필요없으며 요소 A[j]는 V가 가지는 위치이다. 순환이 진행될 때 A[j]가 V에 대한 위치를 지정하며 첫번째 혹은 두번째 항목이 거짓이든 아니든 무관심하다는것을 관찰한다.

목록 9-1. InsertionSort()함수

```
void InsertionSort(char A[], int n) {
    for(int i = 1; i < n ; ++i) {
        if ( A[i] < A[i-1]) {
            char v = A[i];
            int j = i;
            do {
                A[j] = A[j-1];
                --j;
            }while((j > 0) && (A[j-1] > v));
        A[j] = v;
        }
    }
}
```

9.4.2 InsertionSort()의 질

정렬알고리듬을 해석할 때 우리는 일반적으로 정렬에 의해 수행되는 원소비교의 총 회수와 원소복사/값주기의 총 회수에 관심을 두게 된다. InsertionSort()에 대하여 매개 반복자 j에 대한 A[j]의 값은 처음 i개 요소값들중에서 제일 작을 때 요소비교와 복사/값주기의 회수는 최대로 된다(즉 A가 반대로 배렬되였다). 그러므로 i가 1일 때 한번의 요소비교와 3번의 요소복사/값주기가 진행된다. i가 2일 때 기껏해서 2번의 비교와 4번의 요소복사/값주기가 진행된다.

일반적으로 반복자 i상에서 기껏해서 i번의 요소비교와 i+2번의 요소복사/값주기가 InsertionSort()에 의하여 진행된다. n이 목록 A의 요소들의 수라고 하자. 그러므로 총 비교회수는 1+2+…n-1이며 n^2에 비례한다. 이 총수는 InsertionSort()의 최악의 경우에 대한 수행을 반영한다. 그 수행은 요소들의 수의 제곱에 비례하기때문에 그 알고리듬은 최악의 경우 집행이 일치한다고 본다.

InsertionSort()가 무질서한 순서로 된 값들의 목록으로 주어 졌다고 가정하자. 평균반복에 의하여 초기에는 무질서한 순서들의 값들을 가지고 값 V는 현재보조항목의 가운데요소안으로 평균하여 밀기된다. 따라서 무질서하게 배렬된 목록의 n개 요소들에 대한 요소의 비교회수와 복사/값주기회수는 둘 다 최악의 경우수행이 1/2에 비례한다. n^2 /2이 n^2 에 비례하기때문에 무질서한 경우 동작은 역시 일치한다.

InsertionSort()는 이미 배렬된 순서에서 값을 가진 목록의 요소가 주어 질 때마다 오직 한번의 요소비교는 순환당 진행되고 복사/값주기는 하지 않는다. 그러므로 이미 정렬된 목록의 n개 요소들중에서 n-1번의 비교는 진행되고 원소의 복사/값주기는 진행되지 않는다. 이 동작은 InsertionSort()의 가장 리상적인 경우를 표현한다. 그 동작은 요소들의 수에 비례하기때문에 가장 좋은 선형동작을 가진다고 말한다.

대부분의 자료처리전문가들은 정렬하려는 대표적인 목록들이 체계적으로 발생되고 이미 거의 정렬되였다고 생각한다. 목록이 거의 정렬되였다는것을 량적으로 정확히 나타낼수 없지만 거칠게 말한다면 거의 정렬된 목록에서는 매값의 시작위치가 그 목록의 정렬된 판본에서의 그의 위치에 근사하다. 여기서 InsertionSort()에 의해 수행되는 요소비교의 평균회수와 요소의 복사/값주기의 평균회수는 둘 다 n에 비례한다. 이 동작의 특성때문에 InsertionSort()는 대표적으로 자료가 체계적으로 발생되였다는것을 알 때 선택하는 방식이다.

문 제

14. C++에서 배렬들이 값에 의해 넘겨 질수 있는가?
15. 문자들의 배렬을 입력으로 가지고 그것들을 반대순서로 또 다른 배렬에 되돌리는 reverse함수의 본체를 쓰시오. 실례로 아래에서는 I am a good student를 출력하는 정의를 준다.

```
const int N=12;
input [N]='tneduts doog am'I";
output [N]=" " ;
reverse (input,output, N);
cout << output << end1;
```

16. 3개의 형식파라메터들을 가진 **int**형함수 Lessthan()을 쓰시오. 파라메터는 **int**형배렬 a, 배렬에서 요소개수인 **int**형변수 n, **int**형값 V이다. 값 V는 기정적으로는 0으로 선택된다. 함수 Lessthan()은 V보다 작은 항목 a[0], a[1],…, a[n-1]의 요소들의 개수를 돌린다.
17. 파라메터들로서 2개의 옹근수배렬과 배렬의 크기를 가지는 Equal이라는 함수를 쓰시오. 함수 Equal()은 2개의 배렬들이 꼭 같으면 참을 돌린다. 다르면 거짓을 돌린다. 배렬이 꼭 같다는 의미는 같은 순서에 같은 값이 있다는것이다.
18. 파라메터들로서 하나의 옹근수배렬과 배렬의 크기를 가진 IsSorted라는 함수를 쓰시오. 함수 IsSorted()는 그 배렬이 커지는 순서로 정렬되였으면 참을 돌리고 아니면 거짓을 돌린다.
19. 간단하면서도 자주 사용되는 정렬알고리듬은 거품정렬(BubbleSort)이다. 거품정렬은 알고리듬이 응용될 때 가장 작은 항목들이 배렬꼭대기에서 《거품처럼 부글부글 솟아 나오》므로 그렇게 부른다. 그것은 아주 불충분한 알고리듬이다. 기본개념은 이웃한 요소들을 련속 비교하는것이고 그것들이 순서를 벗어 나면 교환한다. 만일 작아 지는 순서로 정렬된 목록이라면(배렬에서 첫 요소가 제일 작다.) 제일 큰(순서로) 요소들이 밑으로 내려 갈 동안 제일 작은 원소가 꼭대기로 《솟아 오른다》. 첫번째를 통과한후 가장 큰 요소는 배렬의 바닥에 놓인다(즉 a[N-1]요소에 놓인다). 두번째로 통과하면 그다음 가장 큰 요소가 정확한 위치에 놓인다(즉 a[N-2]에 놓인다). 표준입력으로부터 하나의 수를 받아 들이는 프로그람을 쓰시오. 이것은 정렬을 위한 배렬의 크기이다.
    란수값클라스 Randint를 사용하여 우연값으로 배렬을 만족시킨 다음 bubblesort알고리듬을 사용하여 그것을 정렬하시오. 배렬크기 4000을 가지고 그 프로그람을 실행하고 8000을 가지고 실행하시오. 두 실행시간에는 어떤 중요한 차이점이 있는가?
20. 본문파일에서 사용한 문자들의 기둥도표를 생성하는 프로그람을 쓰시오. 프로그람은 ASCII문자를 포함한 파일이름을 인용하고 접수한다. 프로그람은 파일을 읽고 문자가 리용된 회수를 센다. 대문자와 소문자는 구별하지 않는다. 그 파일을 처리한후 프로그람은 cout흐름으로 기둥도표를 출력한다.

9.5 용기클라스

앞에서 본 바와 같이 C++는 배렬의 리용에서 큰 제한성이 있다. 즉 함수의 귀환형은 배렬이 될수 있다. 배렬은 값으로 넘겨 질수 없다. 배렬은 값주기대상이 될수 없다. 2개의 다른 방해되는 제한은 배렬의 크기가 상수로 되여야 한다는것과 배렬은 크기를 다시 정할수 없다는것이다.

즉 일단 배렬이 창조되면 요소들의 수를 증가시키거나 감소시킬수 없다. 배렬의 제한성은 변하는 목록

을 리용할수 없게 한다는것이다. 이식할수 없는 표현들을 사용하는것은 개발자들이 자기의 쏘프트웨어의 다중판본을 만들고 지원해야 하기때문에 매우 많은 품을 요구한다. 이 품은 표준본보기서고(STL)를 리용하여 줄일수 있다.

STL의 용기클라스들은 프로그람작성자들이 지정된 목록에 어떤 형의 원소를 넣어야 하는가를 지정할수 있게 하는 일반적인 목록표현들의 모임이다. 용기클라스들은 배렬에 제한을 가지지 않을뿐 아니라 확장될수도 있다. 실례로 첨자검사를 자동적으로 진행하는 전문화된 용기클라스들을 유도할수 있다.

8개의 주요용기클라스들중 6개는 목록을 원소들의 렬로서 취급한다. 이러한 용기들은 deque, list, priority_queue, queue, stack, vector이다. 다른 두 용기클라스들인 map와 set는 목록을 보다 조합적인 방식으로 취급한다. 8개 클라스들에 대한 주요해설을 표 9-1에서 준다.

표 9-1. 주요용기클라스들

용 기	설 명
deque	자기 순서렬에 있는 개개의 요소들에 임의로 접근하게 한다. 더우기 임의로 자기 순서렬의 앞과 뒤에서 삽입 또는 삭제를 할수 있다
list	자기 순서렬에 있는 개개의 요소들에 임의의 순서대로 접근하게 한다. 더우기 임의로 자기 순서렬의 요소를 삽입하거나 삭제할수 있다
priority_queue	접근에 대한 우선권을 제공한다. priority_queue는 가장 높은 우선권을 가지고 있다. 더우기 자기 순서렬의 임의의 곳에서 요소를 삽입 또는 삭제할수 있다
queue	선입선출의 요소접근방법을 제공한다. queue는 자기 순서렬의시작 혹은 끝에 임의로 접근한다. 더우기 임의로 순서렬의 끝에서 삽입, 앞에서 삭제를 할수 있다
stack	후입선출의 요소접근방법을 제공한다. stack는 순서렬의 끝에 임의로 접근한다. 더우기 임의로 자기 순서렬의 뒤에서 삽입, 삭제할수 있다
vector	임의로 자기 순서렬의 개별적인 요소들에 접근한다. 임의의 시각에 vector는 순서렬의 마지막에 요소를 삽입하거나 삭제할수 있다. 다른 곳으로의 삽입 또는 삭제시간은 순서렬의 크기에 비례한다
map	임의로 목록의 개별적인 요소들에 순서대로 접근한다. 유일한 열쇠값은 매개 요소값과 련계된다. 자기 열쇠값에 기초한 요소접근시간은 목록안에서 요소들의 수에 대한 로그에 비례한다
set	임의로 자기 목록안의 개별적인 원소들에 순서대로 접근한다. 자기의 값에 기초한 원소접근시간은 목록안에서 원소들의 수에 대한 로그에 비례한다

priority_queue와, queue, stack클라스들은 다른 용기들을 리용하여 구성(적응)되므로 용기적응기(container adapter) 또는 적응기(adapter)라고 부른다.

STL에 있는 용기클라스들의 실현은 C++클라스본보기구조를 사용한다. 클라스본보기(class template)는 본보기파라메터(template parameter)라고 하는 형식파라메터를 개별적인 형들과 값들을 위한 자리유지자(placeholder)로서 리용하여 클라스가 취할수 있는 일반형태를 서술한다. 본보기로부터 구체적인 클라스를 생성하기 위하여서는 실지로 관계되는 형들과 값들을 본보기이름뒤에서 <>괄호로 넣어 준다. 이 본보기들을 사용하자면 STL에 그 이름을 주면 된다. 다음의 코드토막은 용기클라스본보기

들을 사용하여 생성된 클라스형들인 3개의 항목 A, B, C를 정의한다.

```
deque<int> A(10.1);     //A는 10개의 요소를 가지는데 그 요소들의 값은 다 1이다.
vector<Rational> B(5);  //B는 0/1의 값을 가지는 5개의 요소로 되여 있다.
queue<float> C;         //C는 float형 빈 목록이다.
```

목록 A는 deque<int>형이며 B는 vector<Rational>형이고 목록 C는 queue<float> 형이다. 그 목록들의 요소들은 각각 int, Rational, float형이다. 목록 C는 queue<float>형이다. 그 목록들의 요소들은 각각 int, Rational, float형이다. 필요에 따라 매 목록의 요소형은 용기본보기의 이름뒤에 있는 <>괄호안에 포함된 림시파라메터이다. 구축자파라메터들은 초기요소들의 초기요소들의 값들을 렬거한다. 다음 부분에서는 용기구축자들이 가질수 있는 여러가지 형태들에 대하여 자세히 론의하게 된다.

비록 3개의 용기정의들이 각기 1개의 본보기파라메터를 하나씩 공급한다 하여도 비련계된 용기클라스본보기들은 일반적으로 하나 또는 2개의 파라메터들을 가질수 있다. 첫번째 본보기파라메터는 용기가 가지고 있는 값의 형이다. 만일 두번째 파라메터가 제공된다면 그 파라메터는 목록의 요소를 위한 기억기할당방법을 수행하는 클라스이다.

련결된 용기클라스본보기들은 하나, 둘 또는 3개의 림시파라메터들을 가질수 있으며 두번째와 세번째 파라메터는 생략할수 있다.

첫 파라메터는 용기가 가지고 있는 값의 형이다. 두번째 파라메터는 비교하는 목록요소들에 대한 구조를 완성하는 클라스이다. 세번째 파라메터는 기억기할당파라메터이다. vector클라스본보기는 가장 위력한 목록표현이다.

그러므로 다음용기클라스실례들에서 그것을 리용한다. 이 클라스에 대한 머리부파일은 vector라고 부른다. 기정기억기배당방법은 대부분의 프로그람작성상황에서 진행하기때문에 클라스본보기의 표현 vector는 선택가능한 해당파라메터를 무시한다.

9.6 클라스 vector

vector클라스본보기는 원소들의 목록을 정의하기 위하여 4개의 구축자들을 제공한다.

- 빈 목록을 정의하기 위한 기정구축자
- 존재하는 목록을 복사하기 위한 복사구축자
- 목록의 초기크기를 지정하는 파라메터를 가진 구축자. 원소들은 목록원소형의 기정구축자를 리용하여 초기화된다.
- 2개의 파라메터를 가진 구축자. 첫 파라메터는 목록의 초기크기를 보여 주며 두번째 파라메터는 매 목록원소의 초기값을 보여 준다.

N, M, length객체들에 값들을 넣어 주는 다음의 코드토막이 실행되였다고 가정하자.

```
const int N=20;
const int M=40;
cout <<"size of List to produce;";
int length;
cin >> length;
```

이 객체의 값들은 vector객체들 A, B, C, D, E의 구축에 리용된다.

```
vector<int> A(10); //10개의 int형 벡토르원소
vector<char> B(M);  //40개의char형 벡토르원소
vector<float> C(M*N); //800개의 float형 벡토르원소
vector<int> D(length); // length개만한 int형 벡토르원소
vector<Rational> E(N); //0/1의 값을 가지는 20개의 Rational 벡토르원소
```

5개의 vector객체들은 각각 개별적인 원소들의 목록이다. 목록들의 초기크기들은 파라메터들로부터 구축자들에 들어 온다. 특별히 D목록의 초기크기는 값의 실행시까지도 알려 지지 않는 객체들로부터 들어 온다. 매 목록원소는 어떤 다른 객체처럼 사용될수 있는 객체이다. 초기형들은 특수한 값들로 자기형의 객체들을 자동적으로 초기화하는 구축자들을 가지지 않는다. 목록 A, B, C, D, E의 원소들은 초기화되지 않는다. 그러나 Rational클라스가 표현 0/1을 생성하는 기정구축자를 가지고 있기때문에 E의 원소들은 초기화된다. 다음의 정의를 보시오.

```
Rational r(1,2);
```

벡토르 F의 정의에 r를 리용한다. 이 정의에서 G, H, I의 정의들은 각각 자기 목록원소들에 대한 특수한 초기값을 보여 준다.

```
vector < Rational > F(N, r); // 1/2값을 가지는 20개의 Rational벡토르원소
vector < int > G(10, 1);        // 값이 1인 10개의 int형원소
vector < char > H(M,'h');    // 값이 h인 40개의 char형원소
vector < float > I(M*N, 0);    // 0값을 가지는 800개의 float형벡토르원소
vector < int > J(length, 2);    // 2값을 가지는 length개의 int형벡토르원소
```

vector클라스본보기에서 복사구축자는 2중목록을 만들수 있게 한다. 실례로 R는 매 요소가 1/2로 초기화된 20개의 Rational형목록이며 S는 매 요소가 1로 초기화된 10개의 int형목록이고 T는 매 요소가 2로 초기화된 length개의 int형목록이다.

```
vector <Rational> R(F);  // F가 중복되는 20개의 벡토르
vector <Rational> S(G);  // G가 중복되는 10개의 벡토르
vector <Rational> T(J);    // J가 중복되는 length의 벡토르
```

같은 원소형을 가진 2개의 vector객체들은 각각 다른것을 할당할수 있다. 실례로 다음의 3개의 정의를 보시오.

```
vector<int> U(10.4);
vector<int > V(5.1);
vector<char > W(10,'a');// vector of 5 chars
```

이 벡토르들은

U	4	4	4	4	4	4	4	4	4	4
V	1	1	1	1	1					
W	'a'	'a'	'a'	'a'	'a'	'a'	'a'	'a'	'a'	'a'

과 같이 표시할수 있다. 이 정의들이 주어 지면 다음의 할당이 유효하며 U를 복사하여 V를 만든다. 이 정의들이 주어 지면 다음의 값주기가 정확하며 U를 복사하여 V를 만든다.

V = U; // V는 U의 2중복사이다.

값주기연산자 =는 vector클라스의 성원연산자이다. 그 값주기연산자는 목록들이 같은 크기가 아니라도 원천과 목적객체들이 같은 형의 목록들을 표현할 때마다 정의된다. 만일 필요하다면 값주기연산자는 값주기원천과 같이 원소들을 가지도록 하기 위하여 값주기대상을 재편성한다. 이와 같이 V에 값주기된 다음 3개의 벡토르는 다음과 같이 표시된다.

U	4	4	4	4	4	4	4	4	4	4
V	4	4	4	4	4	4	4	4	4	4
W	'a'	'a'	'a'	'a'	'a'	'a'	'a'	'a'	'a'	'a'

비록 U와 W가 같은 원소들이라고 해도 다음의 값주기는 콤파일되지 않는다.

W=U; // 틀림

값주기는 U와 W가 다른 형이기때문에 콤파일되지 않는다. U는 vector<int>형이며 W는 vector<char>형이다.

주의

구분기호를 정확히 리용해야 한다

다음의 코드토막을 보시오. A는 무슨 형의 객체인가?

vector <int> A[10];

객체 A는 int형의 10개 원소들을 가진 vector<int>형객체가 아니다. 그 리유는 정의에서 10을 리용하였기때문이다. 10은 둥근 괄호가 아니고 각괄호안에 포함되여 있다. 따라서 A는 vector<int>형의 원소들을 가진 크기가 10인 배렬이다. 언제나 정의를 정확히 주어야 한다.
vector클라스본보기의 기정구축자는 빈 목록, 원소가 없는 목록을 창조한다.

vector클라스본보기의 기정생성자는 빈 목록(요소가 없는 목록)을 만든다.

vector <int> X; //빈 int형벡토르

vector <char> Y; //빈 char형벡토르

빈 목록들은 vector값주기 또는 vector성원함수들 insert()와 push_back()의 리용을 통하여 원소들을 얻을수 있다. 이 벡토르성원함수들과 다른 벡토르성원함수들은 표 9-2와 9-3에 서술하였다. 이 2개의 표들의 size_type는 부호 없는 옹근수형이다. reference는 T&로 변환할수 있다. const_reference는 상수 T&로 변환할수 있으며 const_iterator, const_reverse_iterator, iterator, reverse_iterator는 T객체에 대한 실행을 론하는 지적자와 같은 형이다. 이 형들은 vector의 클라스본보기정의에서 선언되였다.

9.6.1 임의로 접근할수 있는 vector의 원소들

vector본보기클라스는 여러개의 성원함수들과 vector를 이루는 원소들에 접근하기 위한 연산자들을 제공한다. 이 성원방법들은 두 종류 즉 임의접근과 순차접근방법들로 묶음화될수 있다. 실례로 10개 원소들을 가진 vector<int>객체 A에서 임의접근방법을 사용할 때 완성된 연산에서는 i와 j가 0부터 9까지의 값을 가지는 i번째와 j번째 원소들에 접근할수 있다.

표 9-2. vector클라스본보기의 일부 성원함수들

클라스본보기	설 명
size_type size() const	vector에서 원소들의 수를 돌려 준다
bool empty()const	vector원소들이 없다면 true, 있다면 false를 돌려 준다
reference front()	vector의 첫번째 원소에 대한 참조값을 돌려 준다
const_reference front() const	vector의 첫번째 원소에 대한 상수참조값을 돌려 준다
reference back()	vector의 마지막원소에 대한 참조값을 돌려 준다
const_reference back() const	vector의 마지막원소에 대한 상수참조값을 돌려 준다
iterator insert (iterator pos, const T &val = T())	vector의 위치 pos에서 val을 복사해 넣고 vector안에서 복사된 위치를 돌려 준다
iterator erase (iterator pos)	Pos위치에서 vector의 원소를 지운다
void pop_back()	vector의 마지막원소를 지운다
void push_back(const T &val)	vector의 마지막원소뒤에 val을 복사해 넣는다
void resize(size_type s, T &val=T())	n을 vector의 현재원소들의 수로 하자. s>n이면 원소들의 수는 이미 있던 원소들뒤에 새 원소들을 s에 추가한다. 새 원소들의 초기값은 val이다. s>n이면 원소들의 수는 vector의 끝으로부터 원소들을 s에서 지워 버린다. s=n이면 동작은 진행되지 않는다
void vector::clear()	vector로부터 모든 원소들을 지운다
void vector::swap(vector<T> &V)	현재 vector와 vector V는 값들을 바꾼다. 이 연산은 일반적으로 원소들의 개별적인 교환보다 훨씬 더 효과적인 것이다
reference at (int i)	i가 정확한 첨자라면 i번째 원소를 돌려 주며 아니라면 례외가 발생한다
const_reference at(int i)	i가 정확한 i번째 원소를 돌려 주며 아니면 례외가 발생한다. 돌려 주는 원소는 수정될수 있다
const_iterator begin()	vector의 첫번째 원소를 가리키는 귀환값을 돌려 준다. 이 반복자에 의해 참조되지 않은 원소들은 수정될수 없다
iterator end()	마지막원소의 다음위치를 지적하는 반복자를 돌려 준다
const_iterator end()	마지막원소의 다음위치를 지적하는 반복자를 돌려 준다. 이 반복자에 의하여 참조해제된 원소들은 고칠수 없다
reverse_iterator rbegin()	vector의 마지막원소를 가리키는 거꿀반복자를 돌려 준다
const_reverse_iterator rbegin()	vector의 마지막원소를 가리키는 거꿀반복자를 돌려 준다. 이 반복자에 의해 참조해제된 원소들은 수정될수 없다
iterator rend()	첫번째 원소의 린접한 앞머리를 가리키는 거꿀반복자를 돌려 준다
const_reverse_iterator rend()	첫번째 원소의 린접한 앞머리를 가리키는 거꿀반복자를 돌려 준다. 이 반복자에 의하여 참조해제된 원소들은 수정될수 없다

표 9-1에서 지적한것처럼 대부분의 용기클라스들은 임의접근방식들을 제공하지 않는다. 대신 그것들은 앞 또는 반대의 순서로 원소들에 대한 접근을 요구하는 련속적인 접근방식들을 제공한다.

임의접근방법들은 첨자연산자 [ ]의 다중정의이다. 첨자연산자는 **const**와 비**const** 벡토르객체를 위하여 다중정의된다. 비**const**참조연산은 참조의 귀환을 수행하며 귀환값은 접근 혹은 수정될수 있다. **const**첨자연산에 대하여 귀환값은 접근될수 있다.

성원첨자연산수는 배렬첨자화와 비슷한 방식으로 동작한다. vector의 매 원소는 자기의 첨자값을 가진다. 첫번째 원소는 첨자값 0을 가지며 두번째 원소는 첨자값 1을 가진다.

프로그람 9-2는 초기값이 모두 0인 A[0], A[1]로 되는 10개 원소 vector A에서 첨자연산자의 리용을 보여 준다.

A의 초기표현은

A	0	0	0	0	0	0	0	0	0	0
	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]

이다. 프로그람 9-2는 A의 개별적인 원소들에 대하여 련속호출과 수정을 진행한다. 이것들의 마지막에는 원소 A[k]의 값을 추출한다. 만일 표준입력흐름으로부터 입력된 값이 88이면 A의 마지막 표현은

A	1	88	0	0	0	0	0	0	0	0
	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]

로 된다.

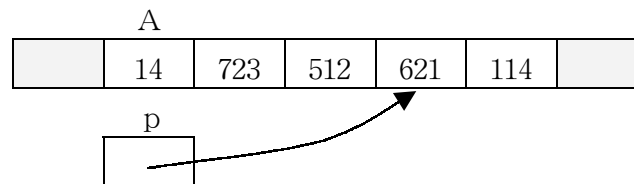
9.6.2 순차접근방법

다른 용기클라스들과의 정합을 위하여 vector는 순차접근방법들도 제공한다. 언급된것처럼 순차접근방법은 주어 진 호출에서 원소들이 참조될수 있는 제한성을 가진다. 그 제한성은 순차접근방법이 쌍방향호출방법인가 한방향호출방법인가에 관계된다. 만일 순차접근방법이 쌍방향이면 련속적인 참조에서 호출된 원소들은 그 목록에서 각각 다른 원소와 린접해야 한다. 만일 순차접근방법이 한방향이라면 호출될수 있는 다음원소는 호출되고 있는 현재원소다음에 곧 나타나는 원소이다.

```
// 프로그람 9-2: 벡토르의 리용법을 소개한다.
#include <iostream>
#include<string>
#include<vector>
using namespace std;
int main()}
    vector<int>A(10,0);//n
    int i = 6;
    int j = 2;
    int k = 1;
```

```
        A[0]=1; // A의 원소 0에 1을 준다.
        A[i]=2; // A의 원소 i에 2을 준다.
        A[j]=A[i]+3;// A의 원소 j에 A의 원소 i에 2를 더한 값을  준다.
        A[j+1]=A[i]+A[0];// A의 원소 j+1에 A의 원소 i와 A의 원소 0을 더한 값을 준다.
        A[A[j]]=4; // A의 원소 A[j]에 4를  준다.
        cout  << A[2]; // 3번째 원소값을 현시한다.
        cin >> A[k]; // k번째 원소값을 입력한다.
        return 0;
}
```

프로그람 9-2. 벡토르첨자연산의 리용법을 설명한다.

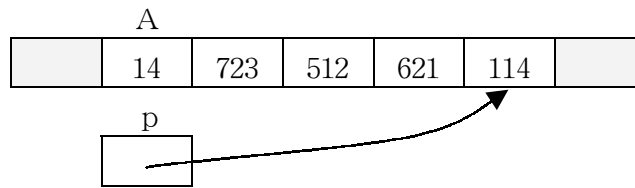
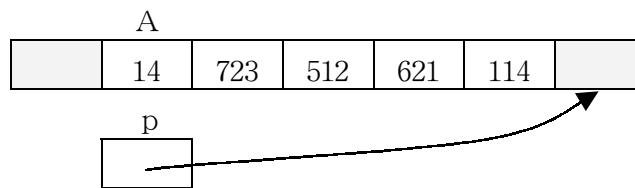
vector가 제공한 순차접근방법들은 다 쌍방향방법들이다. vector순차접근방법들은 반복자를 리용하여 실현된다. 반복자객체의 값은 지적자와 같다. 반복자객체 또는 짧은 반복자는 그 목록의 원소, 그 목록을 이루는 원소들을 둘러 싸는 목록에서 원소들을 지적한다. 이 표현은 다음의 그림에서 서술하는데 반복자 p는 vector<int>형인 5 개의 원소목록 A를 가지고 련계된다. 구체적으로 p는 4번째 원소를 가리킨다. 그림에서 색갈을 칠한 칸은 vector의 원소들을 둘러싼 감시원소(sentinel)이다.

반복자가 가리키는 원소를 참조하기 위하여 단항접두연산자 *를 사용한다. 여기서는 *를 참조해제연산자(dereferencing operator)라고 한다. 이 연산자가 반복자에 적용될 때 반복자가 가리키는 값의 참조를 되돌려 준다. 참조해제연산이 참조귀환을 수행하기때문에 결과값은 볼수도 있고 수정될수도 있다. 실례로 아래의 코드토막을 보시오.

```
cout <<*p<<end1;
*p=821;
cout <<A[3]<<end1;
```

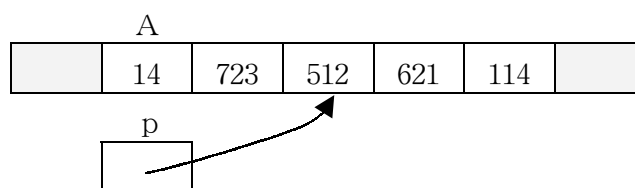
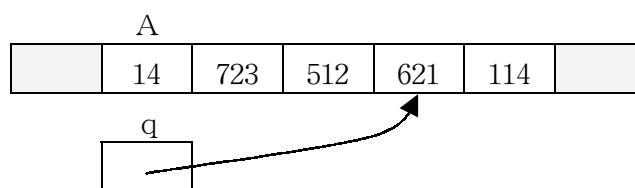
출력결과는 다음과 같다.

```
621
821
```

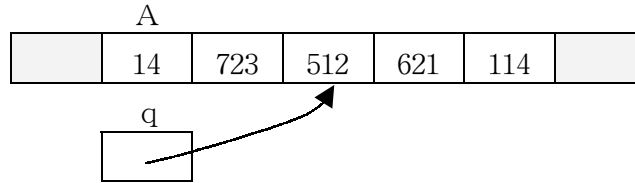
p가 A의 4번째 원소를 가리키기때문에 이러한 출력값이 나온다. 이와 같이 p가 가리키는 값을 수정하는것은 실지로는 A를 수정하는것으로 된다. 구체적으로 A[3]이 수정된다. const객체들의 참조해제연산자는 vector클라스에 의하여 제공된다. const객체들에 대해서 반복자가 가리키는 원소의 값을 호출할수 있지만 그 원소는 수정될수 없다. 증가연산자 ++는 반복자들에 대하여 정의된다. 반복자를 호출하였을 때 증가연산자는 그 목적을 이루는 원소들의 련속에서 다음원소를 가리키는 질문에서 반복자를 갱신한다. 그렇지만 그 목록에 원소들이 더는 없다면 반복자는 대신에 뒤에 달린 감시원소를 가리킨다. 현재

A[3]을 가리키는 반복자 p에 대하여 연산 ++는 A의 마지막원소를 가리키는 p를 생성하는데 이때 A는 A [4] 이다. 객체 A와 p는 아래의 그림으로 묘사하였다.

연산 ++p가 다시 실행되면 p는 감시원소를 가리킬것이다. 이때 A와 p의 상태는 다음의 그림과 같다.

반복자가 쌍방향반복자라면 감소연산자 --도 정의된다. 감소연산자를 호출하면 그 목록에서 앞원소를 가리키는 질문에서 반복자를 발생한다. 실례로 코드토막은 다음과 같다. 감소연산자를 인용하면 반복자는 목록에 있는 앞원소를 가리키게 된다. 실례로 다음의 코드토막에 따라 다시 3번째 위치를 지적할것이다.

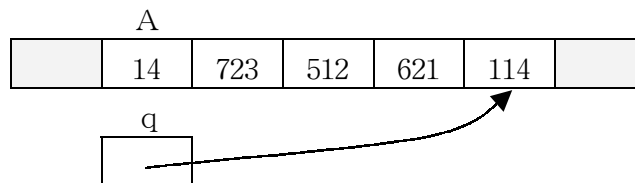
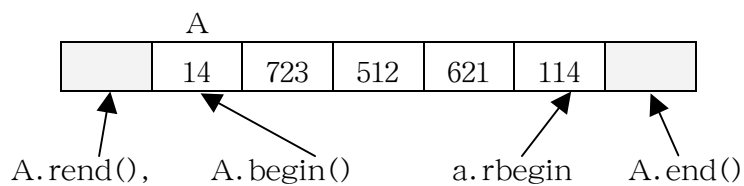
```
--p;
--p;
--p;
```

구체적으로 p는 현재 A[2]를 가리킨다.

만일 앞원소가 목록에 없다면 반복자는 보호성원을 가리킨다. 이 반복자 p는 앞방향반복자이다. 앞방향반복자는 목록의 첫번째 원소로 시작하는 렬로써 목록을 본다. STL은 마지막원소를 가리키는 뒤방향반복자도 제공한다. vector클라스는 이러한 보기도 지원한다. q가 A[]을 가리키는 뒤방향반복자라고 하자.

증가연산

```
++q;
```

는 q가 A[2]를 가리키게 한다.

만일 2개의 감소연산

```
--q;
--q;
```

이 수행되면 q는 A[4]를 지적한다.

vector형클라스는 쌍방향반복자를 되돌리는 4개의 방법 즉 begin(), end(), rbegin(), rend()를 제공한다. 표 9-3에서 지적한것처럼 begin()은 그 목록의 첫번째 원소를 가리키는 반복자를 돌려 준다. end()는 그 목록의 마지막원소다음에 나타나는 감시원소를 가리키는 반복자를 돌려 준다. begin()은 목록의 마지막원소를 가리키는 반복자를 돌려 준다. rend()는 목록의 첫 원소의 앞에 있는 감시원소를 가리키는 반복자를 돌려 준다. vector A에 대한 이 반복자의 표현은 다음의 그림에서 주었다.

rbegin()과 rend()가 돌리는 반복자들은 거꿀형식에서 그 목록을 거꾸로 본다. 이런 반복자들에 대하여 증가연산자 ++는 목록의 앞으로 반복자를 이동하며 감소연산자 --는 목록의 뒤로 반복자를 이동한다. rbegin()과 rend()는 begin()과 end()가 돌려 주는 반복자들과 반대되는 반복자를 돌려 준다.

이 반복자개념들을 증명하기 위하여 먼저 다음의 코드토막을 가지고 초기화된 vector<int>객체 List를 가정한다.

```
vector<int> List(5);
for (int i=0; i < List.size(); ++i){
```

```
        List[i] = 100 +i;
    }
```

따라서 List는 다음의 표현을 가진다.

	100	101	102	103	104	

vector객체들의 반복자의 형들은 vector클라스본보기에서 정의된다. vector<int>클라스가 발생되면 반복자형인 vector<int>::iterator와 vector<int>::reverse_iterator가 정의된다. 이 형이름들은 다루기 불편하므로 반복자들을 자주 사용하는 프로그람들은 문장론적으로 더 변화된 이름을 생성하는 typedef 지령을 사용한다. 실례로 다음의 typedef명령문을 리용하면 vector<int>::iterator 와 vector <int>::reverse_iterator 대신 iterator reverse_iterator를 사용할수 있다.

```
typedef vector<int>::iterator iterator;
typedef vector<int>::reverse_iterator reverse_itertor;
```

례하면 List, iterator, reverse_iterator를 정의한 코드토막은 다음과 같다.

```
iterator p = List.begin();
cout << *P <<"";
++p;
cout << *p <<"";
++P;
cout << *p <<"";
''q;
cout << *p << endl;
reverse_iterator q=List.begin();
cout << *q <<"";
++q;
cout << *q <<"";
++q;
cout << *p <<end1;
''q;
cout << *q <<endl;
```

출력은

```
100 101 102 101
104 103 102 103
```

이다. 반복자 p는 목록에서 첫번째 원소를 벗어 난 위치에서 시작하기때문에 첫 출력행에 현시되는 초기값은 List[0]의 값이다. 이 써넣기후에 p는 두번 증가하는데 매 증가후에 있는 쓰기지령은 p가 현재 가리키는 원소를 현시한다. P가 목록의 끝으로 가는 앞방향반복자이므로 이 조작의 결과 List[1]과 List[2]가 표시된다. 그다음 반복자 p는 감소되고 쓰기지령은 현재 p가 가리키는 원소(List[1])를 현시한다.

두번째 출력행은 List의 마지막원소를 벗어 나 가리키기 시작하는 거꿀반복자 q를 처리한 결과이다. Q가 거꿀반복자이므로 증가연산자는 그 목록의 앞으로 반복자를 전진시키며 감소연산자는 목록의 뒤로 반복자를 가져 온다. 마지막 List원소 List[4]를 현시한 다음 q는 두번 증가하는데 매 증가후에 있는 쓰기지령은 q가 현재 가리키는 원소를 현시한다. 이와 같이 List[3]과 List[4] 가 성과적으로 현시된다. 반복자 q는 감소되며 쓰기지령은 q가 현재 가리키는 (List[3])원소를 현시한다.

다음의 코드토막은 반복자의 대표적인 사용실례이다. 이 토막에서 List의 원소들은 반복자 li를 리용하여 합계된다.

```
int Sum=0;
for (iterator 1i;=List.begin(); li != List.end(); ++li;) {
    Sum = Sum + *li;
}
```

반복자 p처럼 반복자 li도 vectorList의 첫 원소를 지적한다. li가 목록의 마지막원소다음에 즉시 나타나는 감시원소를 가리키지 않는 한 순환은 계속된다. 매 반복마다 실행된 총합에 li가 현재 가리키는 원소의 값을 더한다. 순환검사식의 다음 평가를 위하여 반복자가 증가된다.

9.6.3 vector넘기기

의도에 따라 vector객체들은 다른 형의 객체들처럼 사용될수 있다. 실례로 그것들은 값 또는 참조에 의하여 넘겨 질수 있다. 또한 그것들은 함수에 의하여 되돌려질수 있다. vector객체를 넘기거나 되돌리는 함수문법에는 특수한 표기법이 없다.

목록 9-2에 함수 GetList()와 PutList()가 있다. 이 함수들은 묶음목록들을 처리한 프로그람 9-1을 위하여 개발된 함수들이다.

목록 9-2. 벡토르 GetList(), PutList(), GetValues()

```
void GetList(vector<int> &A){
    int n=0;
    while((n<A,size())&&((in>>A[J])) {
        ++n;
    }
    A.resize(n);
}
void PutList(vector<int> &A) {
    for(int i=0;i<A,size();++i) {
    cout<<A[j]<<end1;
    }
    void }GetValues(vector<int> &A) {
    A.resize (0);
    int Val;
    while (cin >> Val){
```

```
        A.push_back(val){
    }
  }
```

함수 GetList()는 형식파라메터 A를 임명한다. 순환은 A가 다른 값을 기억할수 있는 공간을 가지는 동안 즉 저장값이 있을 동안만 반복된다. 순환이 완성된 후에 vector성원함수 resize()가 호출된다. resize()의 파라메터는 vector의 새 크기이다. A의 크기는 줄어 들것이다. 그러나 resize()는 목록의 크기를 증가하는데도 리용할수 있다. resize()를 취급하는 리유는 이후 사용자들이 객체의 Size()메쏘드를 통하여 원소의 개수를 정확히 관측할수 있도록 하기 위해서이다.

GetValues()함수는 남아 있는 값들을 입력하여 vector의 원소로 한다. GetList()와는 달리 GetValues()는 최대한 많은 원소를 입력하기 위하여 미리 강조되지 않는다. GetList()을 위하여 목록 9-2를 준다. 하나를 선택하는것은 GetValues()이다. 함수 GetValues()는 남아 있는 입력값들을 추출하며 그 vector의 원소들을 만든다. GetList()와 달리 GetValues()는 대부분 원소들의 수를 특별히 추출하기 위하여 preconstrain되지 않는다. GetValues()는 vector성원함수 push_back()를 리용하여 주되는 자기과제를 완성한다. 함수 GetValues()는 파라메터가 빈 목록에서 표현되도록 자기의 형식파라메터 A를 재편성하여 시작한다. 대신에 성원함수 clear()를 사용할수 있을것이다. resize()와 같이 clear()는 표 9-3에 서술한다.

GetValues()의 **while**순환은 val에 대하여 각기 추출된 값에 대하여 반복한다. vector A의 크기는 vector성원함수 push_back()을 리용하여 입력이 진행될 때마다 증가된다. 함수 push_back(목록의 끝에 새 원소를 추가하여 자기의 호출 vector의 크기를 증가시킨다. 새 원소의 값은 push_back()의 파라메터의 값이다. 따라서 A.push_back[val]을 반복실행하여 A가 정확히 설정되는가를 확인한다. 입력된 매값은 목록의 끝에 복사된다. push_back()는 약간 품이 드는 조작이다. 새 원소를 위한 공백은 기억기에서 전체 벡토르의 재배치를 요구한다. 만일 충분한 기억구역이 없다면 례외가 발생한다.

목록 9-3은 vector<**int**>객체들에 대한 추출연산의 다중정의이다. 그 다중정의의 형식은 목록 8-7에서 Rational객체에 대한것과 비슷하다. 그 연산자의 왼쪽연산수는 추출을 수행하는 iostream이며 오른쪽연산수는 갱신될 vector이다. **return**명령문을 제외하고 연산자본체는 GetValues()와 같다. 추출이 보다 큰 추출지령의 부분이 될수 있도록 참조에 의하여 sin을 되돌린다.

목록 9-3. vector추출

```
istream & operator >>(istream &sin, vector<int>v&A) {
    A.resize();
    int val;
    while(sin >> val){
      A.push_back(val);
      }
   return sin;
}
```

다음의 코드토막은 다중정의된 연산자의 리용실례이다.

```
vector<int> List;
cout <<"Enter list of number to be processed;";
cin >> List;
```

표준입력은 아래와 같다.

```
30 4 54
21 6 54
```

함수 GetValues와 vector<int>객체에 대한 추출연산자의 다중정의는 용기클라스가 묶음보다 대규모 프로그람에 유연하다는것을 보여 준다.

List	30	4	54	21	6	54

이 함수와 연산자를 가지고 임의의 크기의 목록을 추출하고 저장할수 있다. 이런 추출은 그것들이 고정된 크기로 되야 하므로 묶음을 사용하여 가능하지 않을것이다.

다음에 장의 앞에서 본 묶음함수 Search()와 류사한 vector함수 Search()에 대하여 서술한다. 첫 파라메터 A는 vector<int>형이고 탐색되는 목록을 표현한다. 두번째 파라메터는 int형 Key이다.

```
int Search(const vector<int> &A, int Key){
    for (int i=0; i < A.size(); i++) {
        if (f[j] == Key)
            return i;
        }
    }return A.size();
}
```

실현은 간단하다. 첨자 i는 목록값을 반복하기 위하여 사용하였다. 만일 그 값을 찾으면 현재첨자를 돌린다. 만일 그 값을 찾지 못하면 목록의 크기가 돌려 진다. 이와 같이 묶음형식의 Search()와 비교할 수 있는 방식으로 동작한다.

경험

일반적인 vector추출

14장에서 본보기함수들과 클라스들을 구체적으로 취급한다. 다음의 코드를 보시오.

```
template<class T>
void GetValues(vector<T> &A){
    A.resize(0);
    T.val;
    while(cin>>val)}
        A.push_book(val);
    }
}
```

template<class T>는 다음의것이 일반적인 형태라는것을 가리킨다. 이 경우 함수의 형이 주어 진다. 이 함수본보기는 실파라메터로 사용된 vector의 특수한 형에 기초하여 각이한 함수들이 동작하도록 한다. 실례로 정의

```
vector<int> X;
vector<Rational> Y;
```

에 의하여 호출

```
Get values(X);
Get values(Y);
```

은 다음의 함수들이 정의되고 호출될수 있게 한다.

```
void Getvalues(vector<int> &A){
    A.resize(0);
    int val;
    while( cin >> val){
        A.push_back(va
    }
}
void GetValues(vector<Rational> &A)}
    A.resize(0);
    Rational val;
    while(cin >> val){
        A.push_back(val);
    }
}
```

로 정의하고 호출한다.

경험

목록원소들의 자동첨자검색

표 9-3에서 지적한것처럼 vector클라스본보기는 그 목록에 첨자값 i를 자기 파라메터로 하는 성원함수 at()를 제공한다. 만일 변수가 유효하면 vector의 i째 원소참조가 되돌려 진다. 유효하지 않다면 례외가 발생한다. 계승을 리용하여 첨자검사를 첨자연산자를 가지고 자동적으로 진행하도록 첨자연산자를 다중정의하는 클라스를 파생한다(13장). 이 동작을 수행하는 본보기클라스 Safevector를 아래에 준다. 용기들을 개발하는 다른 방법은 11, 14장에서 취급된다.

```
template<class T>
class SafeVector:public vector<T>{
    public;
        SafeVector :vector<T>(){}
        SafeVector<int n>:vector<T>(n){}
        SafeVector(int n,T.v):vector<T>(n.v){}
        T&operator[](int i){
            return at[i];
        }
        const T& operator[](int i)const {
            return at[i];
        }
};
```

용기클라스사용의 우월성의 하나는 용기들에 포함된 많은 기초과제들이 STL의 알고리듬서고에서 정의된다는것이다. 대표적으로 이 함수들은 용기의 원소들을 처리하기 위하여 반복자들을 사용한다. 실례로 STL의 알고리듬서고는 search()와 대등한 과제 find()함수를 포함한다. 함수 find()는 3개 파라메터를 요구한다. 첫 2개의 파라메터들은 용기안에서 반복자 p와 q이다. 세번째 파라메터는 값 V이다.

함수 find()는 p에서 시작되고 q에서 끝나는 목록의 원소들속에 v가 있는가를 결정한다. 만일 값V가 목록에 있다면 find()는 목록의 처음 만나는 v위치의 반복자를 돌려 준다. 없으면 find()는 값 9를 돌린다. 앞실레의 vectorList를 리용하여 호출

find(List,begin(),List,end(),54)는 List[2]

는 가리키는 반복자를 돌리며 호출

find(List,begin(),List,end(),9)

는 List.end()의 같은 반복자를 돌린다. List.end()는 9가 목록의 값이 아니기때문에 List.end()가 돌아 간다. 알고리듬보조서고에서 정의된 일부함수들의 목록을 부록 1에 주었다. 그 함수들중 일부를 이 본문에서 개발한 함수대신 리용할수 있다. 그러나 본문에서 개발한 함수는 기초로 되는 알고리듬기술이다른 문제에서 유용하기때문에 가치가 있다. 표준알고리듬함수들을 과제에 리용할 때 검토를 하여야 한다.

9.7 고속정렬

이 장앞에서 론한 함수 InsertionSort()가 비록 평균적인 경우와 가장 좋은 경우의 동작을 가진다해도 최악의 경우와 임의의 경우 동작이 표준으로 된다. 림계시간을 중요시하는 응용프로그람에는 InsertionSort()가 적합치 않을수 있다.

이런 응용프로그람은 QuickSort()방법을 자주 리용한다. QuickSort()는 최적일 때, 최악일 때, 평균일 때 할것 없이 모두 nlogn에 비례한다.이것을 때때로 선형관계라고 한다. QuickSort()의 최악의 경우는 목록에 있는 값들이 역방향으로 정렬된 순서로 있을 때 생긴다. 그러나 이런 경우는 극히 드물다.

QuickSort()방법은 가운데값을 선택하여 시작한다. 그다음 목록은 구획이나 세개의 부분목록에 재배렬된다. 중간부분목록은 값이 중간값인 원소들로 이루어 졌다. 왼쪽부분목록의 원소의 값은 중간값보다 작으며 오른쪽부분목록의 원소의 값은 중간값보다 크다. 부분목록이 이러한 방법으로 구획화되였기때문에 오른쪽부분목록과 왼쪽부분목록은 정렬된 목록을 산생시키기 위하여 종속적으로 정렬된다. 부분목록은 QuickSort()함수를 재귀적으로 호출하여 정렬된다. 실례로 다음의 목록을 정렬해 보자.

B	'Q'	'W'	'E'	'R'	'T'	'Y'	'U'	'I'	'O'	'P'

벡토르 b를 정렬하기 위한 QuickSort()함수는 아래와 같다.

QuickSort(B, O, B.Size()-1);

만일 'P'가 중간값이라면 구획은 다음과 같은 방법으로 A를 재배치할것이다.

A	'I'	'O'	'E'	'P'	'T'	'Y'	'U'	'R'	'W'	'Q'

A[0]에서 A[2]까지를 왼쪽부분목록으로, A[4]와 A[9]를 오른쪽부분목록으로 분리할수 있다. 전체목록에 대해서 이와 같이 정렬할수 있다. 중간원소값을 잘 선택한다면 왼쪽, 오른쪽부분목록은 둘다 n/2개의 원소로 구성된다. 즉 이 구획방법은 수학적으로는 선형산수라고 볼수 있다.

목록 9-4는 머리부파일 qsort.h이다. 머리부파일에는 함수 QuickSort(), Pivot(), Partition(), Swap()에 대한 원형이 들어 있다.목록 9-5는 QuickSort()방법을 보여주기 위한 qsort.cpp이다.

목록 9-4. QuickSort()에 대한 머리부파일 qsorth.h

```
#ifndef QSORT_H
#define QSORT_H
#include <vector>
using namespace std;
void QuickSort(vector<char> &A, int left,int right);
void Pivot(vector<char> &A.int left,int right);
int Partition(vector<char> &A,int left,int right);
void Swap(char &Value1.char &Value2);
#end;f
```

함수 QuickSort()는 처음으로 정렬될 목록이 있는가를 확인한다(마지막 2개 원소들은 정렬되여야 한다). 첨자 left와 right는 QuickSort()의 현재호출에서 정렬되는 목록 A의 왼쪽과 오른쪽부분원소를 지적한다. 만일 목록원소들이 많다면 함수 Pivot()가 먼저 호출된다. Pivot()의 과제는 중간값이 A[left]에 놓이도록 그 목록을 재정렬하는것이다. Pivot()는 A[right]값이 중간값보다 더 작지 않다는 것을 담보하기도 한다. 기초동작에서 Pivot()는 원소 A[left]와 A[right]를 비교한다. A[left]가 A[right]보다 더 크면 원소값은 교환된다(련습에서 우리는 더 유용한 함수 Pivot()를 취급한다).

정렬되는 목록 A가 Pivot()의 호출에 앞서 다음의 표현을 가진다고 가정한다.

A	'Q'	'W'	'E'	'R'	'T'	'Y'	'U'	'I'	'O'	'P'

Pivot()의 호출후에 A는 다음과 같이 표현된다.

A	'P'	'W'	'E'	'R'	'T'	'Y'	'U'	'I'	'O'	'Q'

다음으로 QuickSort()는 left로부터 right범위의 첨자를 가지는 목록 A의 원소를 가진다. 이 과제는 함수 Partition()에 의해 진행된다. Partition()이 3개의 부분목록들을 구성하였다면 현재 중간값을 포함하는 원소의 첨자를 돌린다. 그다음 두번의 재귀적호출을 진행하여 중심원소의 왼쪽과 오른쪽에 대하여 정렬한다. 정렬의 크기는 Partition()호출에 의하여 진행된다. 이 함수는 요점값 A[left]의 복사 Pivot를 만들어서 시작한다. 그다음 두 첨자변수 i와 j를 정의한다. 이 첨자들은 각각 목록의 오른쪽과 왼쪽측면으로부터의 위치를 지적한다.

Patition()의 주요순환은 첨자 i와 j가 서로 어길 때까지 반복하는 do순환이다(이때 왼쪽측면첨자 i의 값은 오른쪽측면첨자 j보다 더 크다).

목록 9-5. qsort.cpp의 QuickSort()함수 실행

```
#include  "qsort.h"
// QuickSort():
void QuickSort(vector<char> &A, int left, int right) {
```

```
    if(left < right) {
        Pivot(A, left, right);
        int k = Partition(A, left, right);
        Quicksort(A, Left, k'1);
        QuickSort(A, k+1, right);
    }
}
// Pivot():
void Pivot(vector<char> &A, int left, int right) {
    if (A[left] > A[right])
        Swap(A[left], A[right]);
}
// Partition():
// A[left]
int Partition(vector<char> &A, int left, int right) {
    char Pivot = A[left];
    int i = left;
    int j = right + 1;
    do {
        do ++i; while (A[i] < Pivot);
        do --j; while (A[j] > Pivot);
        if (i < j) {
            Swap(A[I], A[j]);
        }
    } while ( i < j);
    Swap{A[j], A[left]);
    return j;
}
// Swap():
void Swap(char &Value1, char &Value2) {
    char RememberValue1 = Value1;
    Value1 = Value2;
    Value2 = RememberValue1;
}
```

Do 순환이 반복될 때마다 i는 값이 Pivot와 같은 제일 마지막원소를 찾을 때까지 내부do순환에서 증가된다. 또한 j는 값이 Pivot와 같은 제일 처음원소를 찾을 때까지 내부do순환고리에서 감소된다. Pivot()가 A[left]를 A[left]와 A[right]를 작은 값으로 하고 A[right]를 큰 값으로 하였기때문에 이런 요소들이 반드시 있어야 한다. A[left]와 A[right]는 Partition()의 두 내부순환이 완료되도록 하는

보초병처럼 동작한다. 첨자 i, j가 갱신된후 두 첨자가 어기지 않는다고 결정되면 A[i]와 A[j]의 값은 잘못된 구획에 있으며 교환되여야 한다. 처음 교환되기전의 상태는 다음과 같다.

A	'P'	'W'	'E'	'R'	'T'	'Y'	'U'	'I'	'O'	'Q'
		i							j	

경험

정렬을 어떻게 빨리 할수 있는가?

최악의 경우 InsertionSort()과 QuickSort()는 불과 거의 n개 원소로 된 목록을 정렬하기 위하여 n^2 번의 원소비교와 원소복사/값주기를 한다. 만일 목록이 n개의 서로 다른 값으로 된 원소로 이루어 졌다면 n차례곱개의 가능한 조합의 목록이 있을수 있다. 첫 원소가 비교되면 n차례곱의 절반순서로 정렬된다. 실례로 i번째 원소가 j번째 원소보다 더 작다면 i번째 원소의 값은 j번째 원소값보다 더 앞서게 된다.두번째 비교가 진행되면 최악의 경우에 적어도 n차례곱의 4분의 1만한 순서가 아직도 남아 있게 된다. 세번째 비교후 n차례곱은 아직도 8분의 1정도 남아 있게 된다. 이 경우에 적어도 nlogn의 비교가 필요하다. 그러므로 어떤 비교정렬알고리듬도 자기개수만한 목록의 값만큼은 비교동작이 진행되여야 한다.

만일 목록이 n개의 값들로 되여 있다면 값들을 정렬하는데 n번이 가능할수 있다. 첫 원소비교가 진행될 때 n!순서의 절반은 아직 정렬되여야 할 후보들이다. 실례로 i번째 원소가 j번째 원소보다 더 작다면 i번째 원소의 값이 j번째 원소 의 값에 놓이는 순서는 가능하다. 두번째 비교후 나쁜 경우 n!순서의 마지막 n번째에서 동작은 아직 순서대로 정렬되려는 후보이다. n번째 비교후 나쁜 경우 n!순서의 18번째에서 아직도 취급될것을 요구한다. 나쁜 경우의 동작에서 우리는 하나의 후보만이 취급되기 위하여 남아 있기 전에 nlogn번의 비교를 요구한다. 그러므로 정렬알고리듬에 기초한 어떤 비교는 비교회수에서 적어도 선형수학적인 나쁜 경우동작을 가진다.

경험

QuickSort설명문

Quircksort()알고리듬은 C.A.R.Hoare에 의하여 만들어 졌으며 대부분이 분석화된 콤퓨터과학알고리듬일것이다.

교환후 그 상태는

A	'P'	'O'	'E'	'R'	'T'	'Y'	'U'	'I'	'W'	'Q'
		i							j	

이며 그 처리는 반복된다.우리의 실례를 계속하면 교환전의 상태는

A	'P'	'O'	'E'	'R'	'T'	'Y'	'U'	'I'	'W'	'Q'
				i				j		

로 되며 교환후의 상태는 다음과 같다.

A	'P'	'O'	'E'	'I'	'T'	'Y'	'U'	'R'	'W'	'Q'
				i				j		

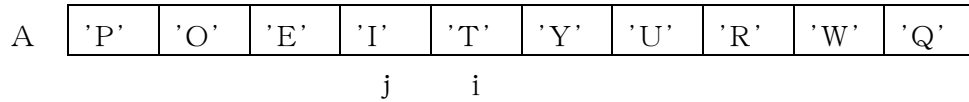
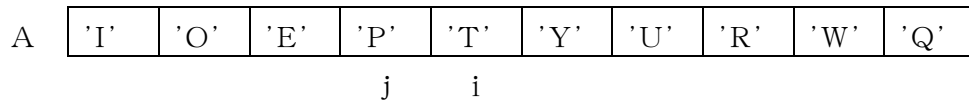
처리는 첨자가 서로 어길 때까지 계속된다. 실례로 바깥do순환의 다음번 반복의 결과는 어긴 첨자등과 함께 다음과 같은 상태로 된다.

첨자가 어길 때 중심값을 제외한 구획이 구축된다. 이때 첨자 j의 값은 오른쪽의 첨자 즉 왼쪽분리부문에 속한 대부분의 원소를 표현한다. 만일 A[j]와 A[left]가 교환되면 A[j]와 A[left]의 교환결과 아래의 상태로 된다.

중간원소와 왼쪽과 오른쪽부분목록을 구성하는것과 함께 함수는 첨자 j를 되돌려 준다. QuickSort()는 2개의 재귀호출에서 중간원소의 첨자를 리용하여 왼쪽과 오른쪽부분목록으로 정렬하는데 실례에서는 목록 A의 첫 4개의 원소와 마지막 5개의 원소로 정렬한다.

그림 9-1은 목록을 정확히 배렬하기 위한 모든 QuickSort()의 호출과정이다.

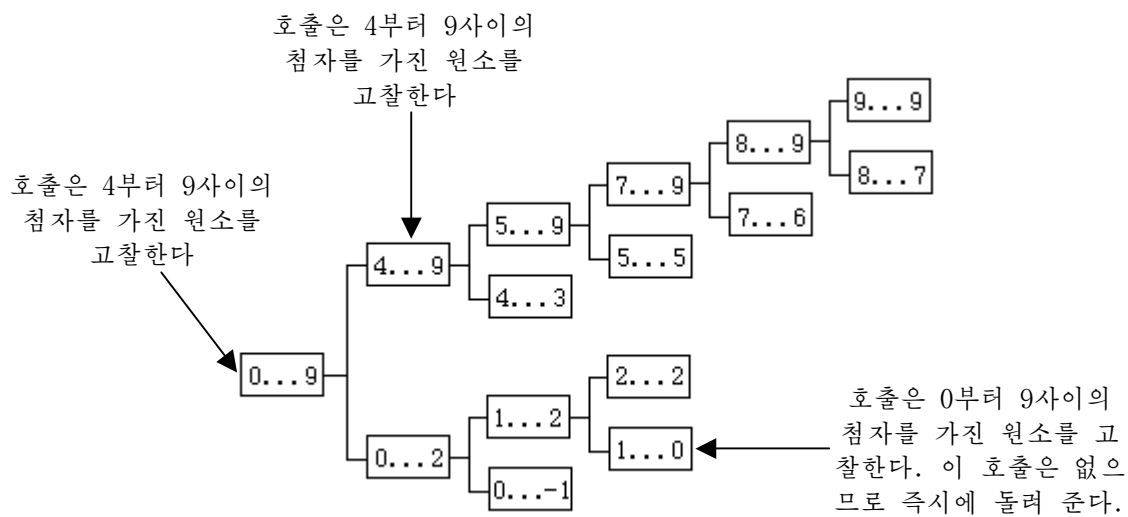



그림 9-1. QuickSort()가 호출될 때 QWERTYUIOP목록에서의 오른쪽과 왼쪽첨자

매 호출의 시작에서 목록의 값묶음은 다음과 같다. 각 항목은 호출시 필요한 left와 right파라메터의 값을 보여 준다. 호출시 목록에 있는 값들의 배렬은 다음과 같다.

0 ... 9: QWERTYUIOP
0 ... 2: IOEPTYURWQ
0 ...-1: EOIPTYURWQ
0 ... 1: EOIPTYURWQ
1 ... 2: EOIPTYURWQ
1 ... 0: EOIPTYURWQ
2 ... 2: EOIPTYURWQ
4 ... 9: EOIPTYURWQ
4 ... 3: EIOPQYURWT

5 ... 9: EIOPQYURWT
5 ... 5: EIOPQYURWT
7 ... 9: EIOPQYURWT
7 ... 6: EIOPQYURWT
8 ... 9: EIOPQYURWT
8 ... 7: EIOPQYURWT
9 ... 9: EIOPQYURWT

9.8 2진탐색

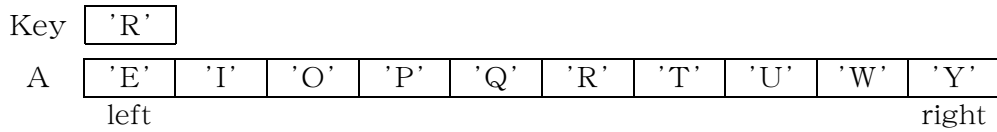
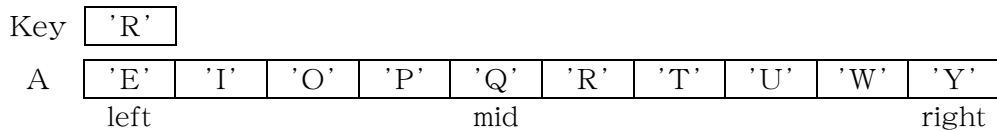
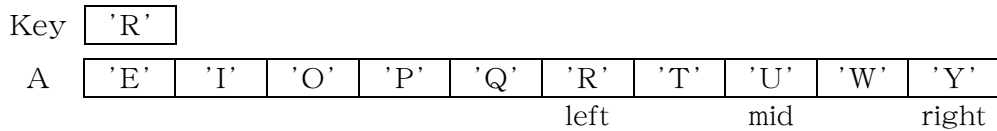
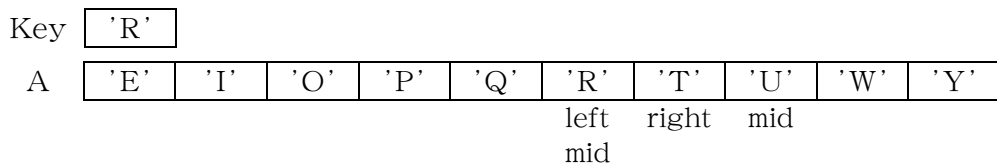
목록이 정렬되여 있을 때 어떤 값이 목록에 있는가를 검색하는 Search()함수보다 좋은 방법이 있다. 실례로 전화책에서 이름을 볼 때 처음부터 시작하지 않고 이름을 찾을 때가지 검사한다. 이름이 목록에서 순서대로 정렬되였다는것을 리용하며 일부 재치 있게 오른쪽페지로 고속으로 뛰여 넘어서 검사를 시작한다. 실례로 전화번호책에서 이름을 찾을 때에는 시작부터 이름을 찾을 때까지 훑어 보지 않는다. 이름이 정렬되여 있다는 사실을 리용하여 오른쪽페지에로 빨리 넘어 가 찾기 시작한다.

목록 9-6에서 준 함수 BinarySearch()는 값 Key를 포함하는 목록 A의 부분을 반복하여 줄이는 검사렬들을 유도한다. 함수 BinarySearch()는 Search()와 같은 변환이 따른다. 만일 Key값이 있으면 BinarySearch()는 검사된 원소의 첨자를 돌려 준다. 만일 Key값이 없으면 BinarySearch()는 A.size()를 돌려 준다. 열쇠값을 포함할수 있는 목록의 부분은 첨자 left 와 right에 의하여 표현된다. if검사에 앞서서 어떤 목록원소는 열쇠값을 포함할수 있다.

목록 9-6. BinarySearch()함수

```
int BinarySearch(vector<char> &A, char Key) {
    int left = 0;
    int right = A.size() - 1;
    while(left <= right) {
    int mid = (left + right) /2;
    if(A[mid] == Key)
        return mid;
    else if(A[mid] < Key)
        left = mid + 1;
    else
        right = mid - 1;
    }
    return A.size();
}
```

그러므로 왼쪽은 0으로 초기화되며 오른쪽은 A.size()-1로 초기화된다. 목록과 열쇠값이 아래의 표

현을 가진다고 가정한다.

while순환은 첨자 left와 right를 갱신하는 검사를 수행한다. 순환은 Key값을 찾든가 열쇠값을 포함하는 목록의 부분이 없다고 결정하였을 때까지 반복된다. while순환의 본체는 객체 mid의 현재 왼쪽과 오른쪽값의 평균을 할당하여 시작한다. 원소 A[mid]가 Key값과 같으면 mid가 귀환된다. 만일 원소 A[mid]가 Key값보다 작으면 목록이 정렬되였기때문에 A[mid]의 왼쪽의 모든 원소들은 Key값보다 모두 작다. 만일 Key값이 목록에 있는것이라면 그것은 A[mid]의오른쪽에 있어야 한다. 그러므로 첨자 left는 mid의 오른쪽린접에 재설정된다. 즉 left는 right+1이 된다. 실례로 Key값은 R인데 R는 A[mid]값보다 더 크며 A[mid]는 'Q'이다. 다음순환시작에서 mid를 갱신한후 객체들은 아래와 같은 그림으로 된다.

만일 원소 A[mid]가 같지 않거나 Key값보다 더 작지 않다면 그것은 Key값보다 더 커야 한다. 목록이 정렬되는데 따라 A[mid]의 오른쪽에 있는 모든 원소들은 Key값보다 역시 크다. 만일 Key값이 목록에 있으면 A[mid]의 왼쪽에 가깝다. 만일 A[mid]가 같지도 작지도 않다면 Key값도 클것이다. 목록이 정렬되여 있으므로 A[mid]의 오른쪽원소들은 모두 Key값보다 더 크다. Key값이 목록에 있다면 A[mid]의 왼쪽에 있을것이다. 그러므로 첨자 right가 mid의 왼쪽에 곧 나타나는것으로 재설정하는 경우 right는 left-1이 된다. 앞서 해설한 상태에서는 Key값이 'R'이고 A[mid]보다 작으며 A[mid]는 '0'이다. 다음순환의 시작에로 mid를 갱신한후에 객체 E는 다음의 표현을 가진다.

만일 A[mid]가 Key값보다 작지도 않고 같지도 않다면 이때 A[mid]는 Key값보다 더 큰것으로 된다. 목록이 정렬됨으로써 A[mid]의 오른쪽에 있는 모든 원소들은 Key값보다 크게 된다. 만일 Key값이 목록안에 있다면 A[mid]의 오른쪽값이 Key값으로 된다. 그러므로 이러한 경우에 첨자 right는 mid의 왼쪽에 놓이게 되며 그리하여 right는 left-1로 된다. 앞에서 보여 준바와 같이 Key의 값은 'R'이며 A[mid]의 값보다 작게 된다. 또한 그것은 'U'로 된다. 반복자의 시작점에서 mid가 갱신된후에 객체들은 다음의 그림과 같이 표시된다.

여기서 Key()와 A[mid]의 값은 다 같은것으로 검사한다. 그리고 함수는 mid의 값을 돌려 준다.

BinarySearch()함수는 n개 원소의 목록을 처리하기 위해 2*log n 번의 비교를 즉시 진행한다는것을 보여 준다. 그 결과 주어 진 Key값이 있는가를 결정하기 위해 BinarySearch()함수는 20번의 비교연산을 진행하여 1000개 원소를 정렬(sort)한다. 1000000개의 목록을 정렬하려면 BinarySearch()함수는 40번 이상의 비교연산을 진행해야 한다.

정보리론에서와 같이 비교연산탐색알고리듬은 log n번의 비교연산을 진행하여야 모든 목록을 정렬할수 있다는것을 보여 준다. STL알고리듬서고는 BinarySearch()함수가 binary_search()함수와 비슷한 동작을 수행한다는것을 정의한다. 이 함수의 구체적인 동작을 보려면 부록 4를 보시오.

9.9 문자렬클라스의 재고찰

다음의 void함수 GetWord()는 입력흐름 cin에서부터 공백이 아닌 문자렬을 추출해 내고 vector <string>객체에 이 값들을 보관한다.

```
void GetWords(vector<string> &List) {
    List.resize(0);
    string s;
    while (cin >> s) {
        List.push_back(s);
    }
}
```

GetWord()함수는 int값을 되돌리는것을 제외하고는 GetValue()함수와 기능이 류사하다. 함수는 string값을 돌려 준다. 만일 표준입력이

```
a list of words
to be read.
```

로 이루어 졌다면

```
vector<string>A;
GetWds(A);
```

는 다음의 방식대로 벡토르 A를 설정한다.

A[0]	a
A[1]	list
A[2]	of
A[3]	words
A[4]	to
A[5]	be
A[6]	read.

문자렬클라스는 문자들의 렬을 가지고 있으므로 클라스용기내에서 보여 줄수 있다. 사실 아래에 쓴

기호는 문자렬객체를 위해 첨자에 짐을 너무 실었다고 볼수 있다. T문자렬객체를 "purple"(자주빛)로서 설명한다고 가정하자. 이미전에 전해 오는 방법으로 그것을 그림으로 표현해야 한다..

t	purple

그러나 이것은 전제로부터 옳게 추론되는것들중 하나를 선택하여 표현해야 한다는것을 의미한다.

t	p	u	r	p	l	e

이 표현들은 일부분들과 서로 다른 원소들을 접근 혹은 수정할수 있으며 지적할수 있다. 례를 들어 그 값주기할당은 t문자렬을 수정하기 위하여 "purple"이라는 문자렬로 표현되였다.

t[1] = 'e'
t[2] = '0'

두번째로 주게 되는 문제(방향)는 문자렬객체 A에서 다시 형성되는 문자렬이라는 규격화된 대형용기내에서 주게 된다.

A[0]	a				
A[1]	l	i	s	t	
A[2]	o	f			
A[3]	w	o	r	d	s
A[4]	t	o			
A[5]	b	e			
A[6]	r	e	a	d	.

같은 방법으로 T문자렬이라는 지적자로 표현되는것들중 하나를 선택하는것과 마찬가지로 개별적인 문자들을 앞에서 표현했던 지적자의 개개의 문자렬목록에서 가까이 할수도 있고 수정할수 있다. 여기서 참조로 되는것은 특히 문자에서 특수한 A문자렬에서 두번째로 사용된 첨자로 선택된 특별한 문자렬로부터 참조하여 두번째 첨자문자렬참조에서 특수한 문자렬을 선택하여 문자렬을 적응할수 있다. 례를 들면 A[i][j]는 A의 i번째 문자렬에서 j번째 문자를 표현한다. 따라서 A는 2차원목록으로 나타낼수 있다. 이에 대해서는 다음절에서 더 구체적으로 보겠다.

다음의 코드토막에서는 겹친 순환구조를 리용하여 A에서 문자 o의 발생회수를 계수한다.

```
int ocount=0;
for (int i=0;i<a.size();++i){
    for (int j=0;j<a[i].size();++j){
        if (a[i][j] =='0'){
            ++ocount;
        }
    }
}
```

바깥for순환은 A의 매 문자렬에 대하여 한번만 반복된다. 매 문자렬 A[i]에 대하여 아낙for순환은 A[i]의 매 문자에 관하여 한번 반복된다. 이 과정에 문자 A[i][j]와 '0'이 비교된다. 만일 그 값이 같으면 account값이 증가된다.

문 제

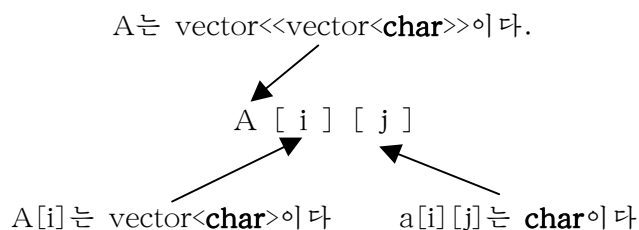
21. int형함수 lessthan()을 3개의 형식파라메터들로써 작성하시오. 즉 옹근수벡토르 a, 벡토르의 원소개수인 int형 n, int형값 v로 작성하시오. 값 v는 기정값 0 에 의하여 선택적으로 쓰일수 있다. 함수 LessThen()은 v보다 작은 벡토르 a[0], a[1], …, a[n-1]의 원소들의 개수를 돌려 준다.
22. 류점수값과 벡토르의 크기를 파라메터로 하는 함수 Equal()을 작성하시오. 이 함수는 두 벡토르가 같다면 true를 돌려 주며 그렇지 않으면 false를 돌려 준다. 벡토르가 같다는것은 같은 순서로 같은 값을 가지고 있다는것을 의미한다.
23. 파라메터로서 벡토르의 크기와 옹근수벡토르를 가지는 IsSorted()를 작성하시오.
24. 간단한것으로 하여 자주 리용되는 정렬알고리듬은 거품정렬(BubbleSort)알고리듬이다. 거품정렬이라는 말은 이 알고리듬이 수행될 때 배렬의 제일 작은 요소가 배렬의 꼭대기에 《거품처럼 솟아 오른다.》는데서부터 유래되였다. 이 알고리듬은 불충분한 알고리듬이다. 기본방법은 이웃한 원소들을 비교하여 그것들을 서로 교환하는것이다. 만일 감소하는 순서로 목록을 정렬한다면 제일 작은 요소가 제일 먼저 놓인다. 제일 큰 요소는 제일 마지막에 놓이게 된다. 두번째 정렬과정이 진행되면 두번째로 큰 요소가 정렬되게 된다.

    표준입력으로부터 묶음의 수를 입력하는 프로그람을 작성하시오. 란수의 클라스 Randint를 리용하여 벡토르를 우연수값으로 채워 놓고 거품정렬알고리듬을 리용하여 정렬하시오. 4천개의 벡토르크기를 가진 프로그람을 실행해 보시오. 또한 8천개의 요소에 대해서도 프로그람을 실행해 보시오. 두 경우에 실행시간은 어떻게 차이나는가?
25. 본문파일에서 리용되는 문자의 기둥도표를 만드는 프로그람을 작성하시오.이 프로그람은 ASCII본문을 포함하는 파일이름을 입력한다. 또한 파일을 읽고 문자의 출현회수를 계산하다. 대소문자는 구별하지 않는다. 파일을 처리한후 프로그람은 cout흐름에 기둥도표를 출력해야 한다.

9.10 2차원목록에서 단어찾기

2차원목록을 고찰해 보자. 벡토르에 대한 벡토르로서 이러한 목록을 표현할수 있다. 이러한 목록은 다음의 명령으로 정의할수 있다.

```
vector< vector<char> >A;
```

A문자는 다음과 같은 객체형의 형태를 묘사한다.

정의에서 기정구축자를 리용하기때문에 A는 빈 목록이다. 그러나 함수 resize()를 리용하여 A의 요

소수를 설정할수 있다. 실례로 13개의 vector<char>요소를 가지는 목록 A를 정의해 보자.

경험

여러번 리용된 <,>기호

프로그람을 번역하는 공정은 사전식분석과 류사하다. 프로그람의 개별적요소들이 여기서 결정된다. 실례로 ==가 나타나면 이 문자들을 두개의 값주기연산자로서가 아니라 같기연산자로 취급한다. 마찬가지로 >>가 나타나면 콤파일러는 이 련속적인 각괄호들을 2개의 개별적인 요소로서 취급하는것이 아니라 하나의 요소로서(즉 삽입연산자로서) 그것들을 묶어 준다. 이러한 자동적인 묶어주기에 의하여 다음과 같은 명령문은 성과적으로 콤파일되지 못한다.

```
vector<vector<int>>A;
```

콤파일러는 이 코드에 대하여 오유통보를 발생시키며 따라서 아래와 같이 다시 써야 한다.

```
vector< vector<int> >A;
```

이 기호사이에 공백을 사용하면 콤파일러는 괄호들을 서로 다른 요소로 취급한다.

다음의 명령문은 A를 재설정하여 13개의 요소를 나타나게 한다.

```
A.resize(13);  //A는 13개의 vector<char>객체를 가지고 있다.
```

명령문에서 resize()는 vector<vector<char>>클라스의 성원이다. 문자의 개수를 설정하기 위하여 vector<char>성원함수 resize()를 리용한다. 실례로 다음의 코드토막은 A의 B개 매 원소들이 9개 문자로 표현되도록 한다. 9개의 요소를 가지는 13개의 목록을 만든다.

```
for (int i=0;i<13;++i){
    A[i].resize(9);
}
```

이 vector<vector<char>>구조체는 2차원문자목록에 어떤 단어가 숨어 있는가 하는 수수께끼를 해결하기 위한 정보를 표현하는데 리용될수 있다. 실례로 Emily, Hannah, James와 Zachary라는 이름을 그림 9-2에 주어 진 문자들중에서 찾을수 있는가? 이름들은 바로 혹은 반대순서로 수평, 수직,대각선상에 위치할수 있다. 목록 9-7에 2차원목록에서 주어 진 단어를 탐색하는 Puzzlesearch()를 정의하였다. 숨겨 진 단어를 더 쉽게 찾을수 있도록 2차원목록은 빈 문자로 둘러 싸여 있다. 실례로 13개의 행과 9개의 렬로 이루어 진 vector<vector<char>>객체가 그림 9-2의 puzzle구체례를 표현하려 하였다면 일부 다른 전문용어들도 쓸모 있게 개선할것이다. 객체는 그림 9-3처럼 표현할수 있다.

vector<vector<char>>는 2차원목록으로 볼수 있다는것을 고려하면 목록을 문자표로 귀착시키는것이 편리하다. T의 요소는 표에서 한개 행이다. T는 렬의 모임으로 볼수 있는데 여기서 j번째 렬은 여러개 행의 j번째 요소로 구성된다. 함수 Puzzlesearch()는 단어의 시작위치를 알아 본다. 단어는 8개 방향으로 처리된다. 행과 렬의 첨자가 요소와 요소를 변경할수 있게 하는 방법은 서로 다르다. 실례로 한개 요소에서 오른쪽앞으로 나가기 위해서 행첨자는 변화되지 않으며 렬첨자가 하나씩 증가한다.

```
A N N E J Z E
S J A M I J N
Y L Z I Q W N
E R A L U V A
R S C Y N N O
D H H L K F J
U J A M E S L
A A R N F H I
S C Y D N D O
U K D Z A A R
A S D F G Q H
```

그림 9-2. 2차원배렬에서 문자들의 탐색

T[0]	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'
T[1]	'\0'	'A'	'N'	'N'	'E'	'J'	'Z'	'E'
T[2]	'\0'	'S'	'J'	'A'	'M'	'I'	'J'	'N'
T[3]	'\0'	'Y'	'L'	'Z'	'I'	'Q'	'W'	'N'
T[4]	'\0'	'E'	'R'	'A'	'L'	'U'	'V'	'A'
T[5]	'\0'	'R'	'S'	'C'	'Y'	'N'	'N'	'O'
T[6]	'\0'	'D'	'H'	'H'	'L'	'K'	'F'	'J'
T[7]	'\0'	'U'	'J'	'A'	'M'	'E'	'S'	'L'
T[8]	'\0'	'A'	'A'	'R'	'N'	'F'	'H'	'I'
T[9]	'\0'	'S'	'C'	'Y'	'D'	'N'	'D'	'O'
T[10]	'\0'	'U'	'K'	'D'	'Z'	'A'	'A'	'R'
T[11]	'\0'	'A'	'S'	'D'	'F'	'G'	'Q'	'H'

그림 9-3. 수수께끼를 탐색하기 위한 목록

이와 같이 대각선방향에서도 행과 렬첨자들은 하나씩 증가해야 한다. 매 시작위치에서 **if-else-if**구조는 다른 방향으로 **if**문을 실행하게 한다. 만일 단어를 찾으면 적당한 통보가 현시되며 함수는 이 값을 되돌린다. 또한 새로운 시작요소에 의해 동작이 계속된다. 동작은 가능한 시작요소가 더이상 없거나 단어가 발견되면 완료된다. 후에 나타난다면 적당한 통보가 현시된다. 구체적인 시작위치와 방향을 검사하는 함수는 CheckWord이다. 이 함수는 목록 9-8에서 보여 준다. 이 함수는 6개의 파라메터를 가지고 있다. 첫 파라메터는 탐색하여야 할 표 T이다. 두번째와 세번째 파라메터는 표에서 문자들을 참조하기 위한 행과 렬의 첨자번호 i, j이다. 이 값은 탐색하기 위한 시작위치로 된다. 4번째 파라메터 Word는 단어를 표현하는 문자렬이다. 5번째와 6번째 파라메터 RowOffset, ColOffset는 숨겨 진 단어를 탐색하는 표에서 다음문자를 호출하기 위하여 첨자들이 얼마만큼 증가되여야 하는가를 나타낸다.

주어 진 방향에 대한 실제적인 검사는 간단하다. 현재행첨자 row와 렬첨자 col은 i, j로 초기화되여 있다. Word에서 매 문자는 변화될수 있다. 현재 Word의 문자 Word[k]와 표 T의 T[row][col]이 서로 다르다면 단어의 위치를 추측하는것은 틀리게 되며 함수는 **false**를 되돌린다. Word와 T의 문자가 다르면 첨자 row와 col은 적당한 값으로 증가되여 T의 새 문자에 대한 첨자를 형성하게 된다. **for**순환이 모든 문자를 정합하면 숨겨 진 단어가 발견되며 함수는 **true**를 되돌린다.

목록 9-7. puzzle.cpp에서 puzzlesearch()함수

```
//PuzzleSearch( ):표 T에서 단어를 검색
void PuzzleSearch(const vector< vector<char> > &T,
    const string &Word) {
    for (int i =1;i<T.size()-1;++i ){
        for(int j=1;j<T[i].size() - 1;++j){
            if(CheckWord(T,i,j,Word,0,1)){
                cout<< Word << " is at " << i << "," << j
                  << " going horizontally right" <<endl;
```

```
                return;
            }
            else if (CheckWord(T,i,j,Word,0,-1)){
                cout<< Word << "is at" << i << "," <<j
                  << " going horizontally left" <<end1;
                return;
            }
            else if (CheckWord(T,i,j,Word,0,-1)){
                cout<< Word << "is at"<< i <<","<<j
                  << "going vertically down"<<end1;
                return;
            }
            else if (CheckWord(T,i,j,Word,0,-1)){
                cout<< Word << "is at"<< i <<","<<j
                  << " going vertically up"<<end1;
                return;
            }
            else if (CheckWord(T,i,j,Word,0,-1)){
                cout<< Word << "is at"<< i <<","<<j
                  << " going diagonally right and down"<<end1;
            return;
            }
            else if (CheckWord(T,i,j,Word,0,-1)){
                cout<< Word << "is at"<<i <<","<<j
                  << " going diagonally left and down"<<end1;
                return;
            }
            else if (CheckWord(T,i,j,Word,0,-1)){
                cout<< Word << "is at" << i << "," <<j
                  << " going diagonally right and up" <<end1;
                return;
            }
            else if (CheckWord(T,i,j,Word,0,-1)){
                cout<< Word << "is at" << i << "," <<j
                  << "going diagonally left and up" << end1;
                return;
            }
        }
    }
```

```
        cout << Word << " is not in the Puzzle" << endl;
        return;
    }
```

알아맞추기탐색표를 초기화하고 사용자가 일련의 탐색을 지정할수 있게 하는 함수 main()을 목록 9-9에 제시한다. 이 함수는 어떤 확인도 진행하지 않는다.

목록 9-8. 함수 puzzle.cpp()의 checkword()

```
//checkword() :시작점 [i][j]에서부터 단어를 찾아 본다.
bool CheckWord(const vector< vector<char> > &T,int i,
  int j,const string &word,int rowoffset, int coloffset){
    int row = i;
    int col =j;
    for (int k=0; k<Word.size(); ++k){
        if (Word[k] != T[row][col])
            return false;
        else{
            row+=Rowoffset;
            col+=Coloffset;
        }
    }
    return true;
}
```

9.11 미로걷기유희

대중유희의 한가지는 시작위치에서 끝위치까지의 통로를 찾는 방법으로 미로를 횡단하는 유희이다. 미로횡단(maze traversal)문제는 흥미 있는 유희일뿐아니라 로보트의 운동이나 콤퓨터소편에서 회로요소들의 호상련결과 같은 중요한 문제들을 야기시키고 있다. 미로실례를 그림 9-4에 주었다. 여기서 시작점은 왼쪽웃구석의 작은 4각형이며 끝나는 점은 오른쪽아래구석의 작은 4각형이다. 어두운 부분은 벽을 의미하며 밝은 부분은 복도이다.

목록 9-9. puzzle.cpp에서 main()함수

```
//main():알아맞추기탐색유희를 관리한다.
int main(){
    string Filename;
    cout << "enter puzzle table filename: ";
```

```
    cin >> Filename;
    ifstream fin(Filename.c_str());
    vector< vector<char> > Table(1);
    while (fin >> s){
        vector<char> v(s.length() + 2);
        v[0]= '\0';
        v[1+s.length()]='\0';
        for (int i=0; i<s.length(); ++i){
            v[i+1]=s[i];
        }
        Table.push_back(v);
    }
    vector<char> null(Table[1].size() + 2, '\0');
    Table[0] = null;
    Table.push_back(null);
    for (int i=1;i< table.size()-1;++i){
        for (int j=1; j<=Table[i].size(); ++j) {
            cout << Table[i][j];
        }
        cout << endl;
    }
    cout << endl;
    cout << "enter your Puzzle search word: ";
    while (cin >> s){
        cout << endl;
        puzzlesearch(Table,s);
        cout << endl;
        cout <<"enter your next search word:";
    }
    return 0;
}
```

이 실레미로는 자유설치벽을 가지고 있지 않다. 자유설치벽이란 주변과 련결되여 있지 않는 벽을 말한다. 표준미로는 오른쪽방략에 의해 해결할수 있다. 오른쪽방략은 벽을 따라 오른쪽으로 미로를 걷는다는것이다. 이렇게 함으로써 종착점에 가닿을수 있다. 이 방략이 경로를 최대한 짧게 해주는것은 아니라는것을 알아야 한다. 성과적으로 유희를 수행하기 위해서 미로의 꼭대기를 북쪽으로, 아래를 남쪽으로 방향을 정할수 있다. 또한 왼쪽면을 서쪽으로, 오른쪽은 동쪽으로 정할수 있다.

그림 9-4에서 미로방황자(mazewanderer)는 동쪽으로 초기방향을 정하고 있다. 오른쪽방략에 의해 처음 8개 단계는 동쪽, 남쪽, 서쪽 등등이다. 가장 좋은 단계는 남쪽인데 방황자는 현재 남쪽방향에 있다. 이러한 상태를 그림 9-5에 보여 준다.

방황자의 다음단계는 서쪽과 남쪽이다. 이 상태를 그림 9-6에 보여 준다. 오른쪽방략에 의하면 방황자는 끝이 난 복도를 반대방향으로 다시 돌아 와야 한다. 그러나 복도에로의 입장이 두번 실현될 때 방황자는 남쪽이 아니라 동쪽으로 마주 섰다. 방황자가 동쪽을 향하기때문에 다음단계는 남쪽으로 향하는것이다. 이러한 상태를 그림 9-7에 보여 준다. 이러한 문제를 해결하기 위하여 프로그람은 3개의 객체를 처리해야 한다. 즉 미로, 방황자, 경로이다. 이 객체들은 클라스형객체로 표현된다. 이에 대응한 클라스들의 이름은 Maze, Wanderer, Path이다.

클라스 Maze, Wanderer, Path는 행과 렬의 위치자리표를 표현하도록 하는 Location클라스를 리용한다. Location클라스의 정의는 목록 9-10에 보여 준다. 이 클라스는 같기연산자 ==, 안같기연산자 !=, 그리고 입력연산자 >>를 다중정의하여 실행된다. Location서고의 정의와 실행은 보다 간단하다.

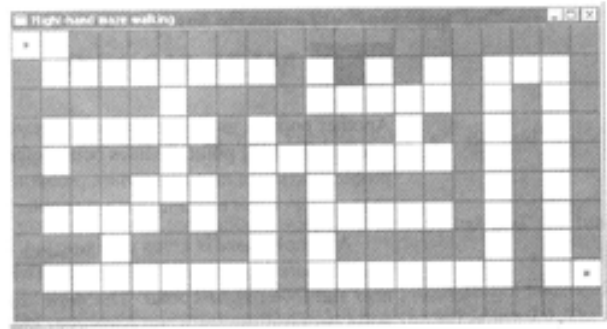



그림 9-4. 입력파일 mymaze.dat와 관련된 표준미로그림

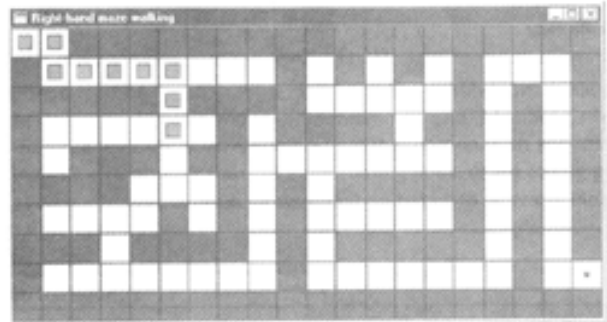



그림 9-5. 방황자의 첫 8개 단계를 묘사한다

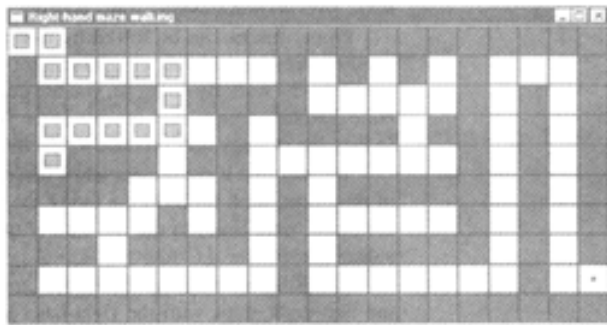



그림 9-6. 방황자의 첫 13개 단계를 묘사한다

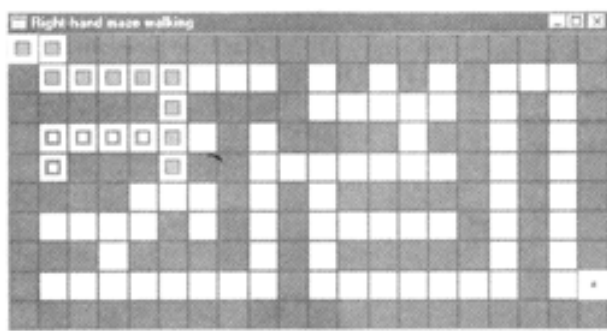



그림 9-7. 방황자의 첫 19개 단계를 묘사한다

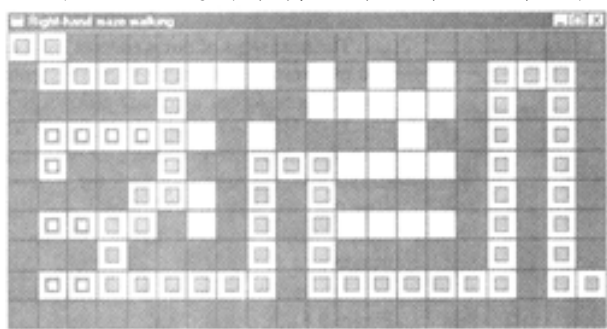



그림 9-8. 방황자의 완성된 경로를 묘사한다

목록 9-10.   location.h머리부파일에서 Location클라스의 정의

```
class location{
    public:
        //constructor
        Location(int r=0,int c=0);
        //inspectors
        int GetRow() const;
        int GetColumn() const;
        //mutators
        void SetLocation(const location&p);
        void SetLocation(intr,intc);
        void SetRow(int r);
        void SetColumn(int c);
    private:
        //Data members
        int Row;
        int Column;
};
```

9.11.1 클라스 MAZE의 명세부

Maze클라스의 중요한 부분은 쌍방향살창으로 된 미로를 표현하는 자료성원 Grid이다. 이 성원은 미로의 매 위치를 기록한다. 가능한 위치상태를 표현하기 위해 렬거형 Status를 리용한다.

```
enum Status {Free,Obstacle,Start,Finish,OutOfBounds};
```

기호상수 Free, obstacle, start, Finish는 미로의 가능한 위치를 표현하며 Status는 상수 outofBounds를 가진다. 상수 outofBounds는 미로부분이 아닌 위치이다. Status형에 의해 vector<vector< Status>>형의 Grid를 정의할수 있다. Grid[r][c]는 미로의 r번째 행에서 c번째 점을 나타낸다. Maze메쏘드를 수행하기 위한 함수들은 다음과 같다.

- Maze(istream&sin=cin) : 입력흐름 sin에 포함된 미로를 지적하는 표현을 초기화하는 기정 구축자.
- Maze(int r,int c) : 크기 r 와 c를 가진 우연적인 미로를 창조하는 구축자.
- Getstatus(const location&p) : 미로의 위치 p상태를 되돌리는 검토자. 만일 이 위치가 미로부분이 아니라면 OutOfBounds값이 되돌려 진다.

목록 9-11.   location.cpp의 원천파일에서 Location클라스의 실현부

```
#include "location.h"
//구축자
location::location(int r,int c){
    setlocation(r,c);
```

```
}
//검토자
int location::GetColumn() const{
    return Column;
}
//변이자
void Location::SetLocation(const Location &p){
    SetRow(p.GetRow());
    SetColumn(P.GetColumn());
}
void Location::SetLocation(int r,int c){
    SetRow(r);
    SetColumn(c);
}
void Location::SetLocation(int r){
    Row=r;
}
void Location::SetLocation(int c){
    Column=c;
}
//==:위치보조연산자
bool operator==(const Location &p,const Location &q){
    return (p.GetRow()==q.GetRow())
    &&(p.GetColumn()==q.GetColumn());
}
//!=:위치보조연산자
    bool operator!=(const Location &p,const location &q){
    return!(p == q);
}
//>>:위치보조연산자
istream& operator >>(istream &sin,Location &p){
    int r;
    int c;
    sin >> r >>c;
    p=Location(r,c);
    return sin;
}
```

- setstatus(const location&p,status&s) : 미로의 위치 p상태를 s로 설정하는 변이자

- Getstart()and Getfinish() : 미로의 시작점과 끝점을 되돌리는 검토자
- Getnumberrow()과Getnumbercolumn() : 미로의 크기를 되돌려 주는 검토자

Maze자료성원은 다음과 같다.

- startpos : 시작해야 할 지점의 위치
- finishpos : 완료된 지점의 위치

이러한 두개의 자료 Location성원은 자료성원 Grid와 류사하다. Maze클라스의 정의는 목록 9-12에 보여 준다.

목록 9-12. maze.h머리부파일에서 Maze클라스의 정의

```
class Maze{
    public:
        //구축자:파일에서부터 미로를 추출한다.
        Maze(istream&sin=cin);
        //구축자:c에 의해서 r크기를 가진 우연미로발생
        Maze(int r, int c);
        //검토자
        status GetStatus (const Location &p)const;
        location GetStart() const;
        location GetFinish() const;
        int GetNumberRows() const;
        int GetNumberColumn() const;
    protected:
        //변이자
        void SetStatus(const Location &p, const Status &s);
    private:
        //자료성원
        vector< vector <status> > Grid;
        Location StartPos;
        Location FinishPos;
};
```

Maze정의는 자료성원을 **private**형으로 하여 정보은폐를 하게 한다. 또한 성원함수 SetStatus()를 **protected**로 만든다. SetStatus()의 호출제한성은 미로를 반영하며 일단 구축되면 변화되지 않는다. 이 함수를 **protected**형으로 함으로써 미로를 변화시킬수 없다. 그러나 **protected**성원의 호출제한은 Maze에서 파생된 클라스가 이 성원을 리용하게 한다.

9.11.2 클라스 wanderer의 명세부

Wanderer클라스를 개발하는데서 Maze객체와 방황자가 호상작용하는것을 피해야 한다. 이렇게 하

기 위해 Wanderer클라스를 재리용할수 있다. 목적을 실행하기 위해 일반목적의 방황자를 개발하여야 한다. 다음의 특성은 방황자가 수행해야 할 특성이다.

- 주의에 대한 상태를 알려 주는 신호를 접수하고 처리하는 능력
- 더 쉬운 방향을 알려 주는 지령을 수행하는 능력

방향을 표현하기 위하여 렬거형 Direction을 리용한다.

```
enum Direction { North, East, South, West } ;
```

4개의 방향을 시계바늘이 돌아 가는 방향으로 고찰해 보자. 형은 direction.h에 정의되여 있다. 이 머리부파일은 역시 함수 clockwise()와 함수 counterclockwise()를 포함한다.

```
Direction Clockwise(const Direction &d);
Direction CounterClockwise(const Direction &d);
```

함수 clockwise()는 파라메터 d를 시계바늘과 같은 방향으로 되돌린다. 실례로 clockwise(north)은 East를 되돌린다. 함수 counterclockwise()는 파라메터가 시계바늘과 같은 방향으로 되기전에 그 방향을 되돌린다. 실례로 counterclockwise()는 South를 되돌린다. clockwise()와 counterclockwise()는 련습에서 구체적으로 보기로 하자. 다음의 Wanderer클라스의 성원메쏘드들을 보자.

- Wanderer(**const** location &p=location(0,0),**const** direction &d=east):위치 p가 방향 d의 정면에 있는 방황자를 구축하는 구축자.
- Getdirection():방황자가 현재 정면으로 되여 있는가를 되돌리는 검토자.
- Setdirection(**const** direction &d) :방황자가 방향 d의 정면에 놓이도록 하는 변이자.
- Getlocation():방황자의 위치를 되돌리는 검토자.
- Looknorth(**const** status &s):방황자의 북쪽위치에 대한 상태를 통보 s로 알려 주는 촉진자. 즉 북쪽의 위치상태는 north이다.
- Looksouth(**const** status &s):방황자의 남쪽위치에 대한 상태를 통보 s로 알려 주는 촉진자. 즉 남쪽의 위치상태는 south이다.
- Lookeast(**const** status &s):방황자의 동쪽위치에 대한 상태를 통보 s로 알려 주는 촉진자. 즉 동쪽의 위치상태는 east이다.
- Lookwest(**const** status &s):방황자의 서쪽위치에 대한 상태를 통보 s로 알려 주는 촉진자. 즉 서쪽의 위치상태는 west이다.
- Movenorth() :현재위치의 남쪽에 방황자의 위치를 옮기는 동작을 수행하는 변이자.
- Movesouth() :현재 남쪽의 위치에서 위치를 옮기는 동작을 수행하는 변이자. 이 성원을 미리 호출한다면 적어도 현재위치에 있는 다음에 방황자는 함수와 같은 기능을 가진 함수에 의해 만들어 지며 Obstacle이나 OutofBounds통보를 서로 다른 방법으로 얻을수 있다. 방법대로 한다면 south는 남쪽으로 흐르게 된다. 만일 이 방법대로 하지 않는다면 south는 아무런 변화도 없을것이다.
- MoveEast() :현재 동쪽의 위치에서 방황자의 위치를 옮기는 동작을 수행하는 변이자. 이 성원을 미리전에 호출한다면 적어도 현재위치에 있는 다음에 lookeast()를 호출해야 한다. East는 함수와 같은 기능을 가진 함수에 의해 만들어 지며 Obstacle이나 OutofBounds통보를 서로 다른 방법으로 얻을수 있다. 만일 이 방법대로 한다면 east()는 동쪽으로 가게 된다. 이 방법대로

하지 않는다면 east는 아무런 변화도 없을것이다.

- Movewest() :현재 서쪽의 위치에서 방황자의 위치를 옮기는 동작을 수행하는 변이자. 이 성원을 미리전에 호출하자면 적어도 현재위치에 있는 다음에 lookwest()를 호출해야 한다. lookwest()는 함수와 같은 기능을 가진 함수에 의해 만들어 지며 obstacle이나 outofbounds 통보를 서로 다른 방법으로 얻을수 있다. 만일 이 방법대로 하면 west는 서쪽으로 가게 된다. 이 방법대로 하지 않는다면 west는 아무런 변화도 없을것이다.

이 메쏘드들을 실현하자면 여러개의 자료성원들이 요구된다. 현재위치 Currpos와 방황자가 마주하고 있는 현재방향 Currdirection이라는 두 자료성원이 필요하다는것은 분명한 사실이다.

Wanderer클라스의 완전한 정의는 목록 9-13에 주어 져 있다. Wanderer클라스의 정의는 그 클라스가 모든 자료성원들을 비공개로 함으로써 정보은폐를 지원하고 있다는것을 보여 준다. 그러나 성원들은 의뢰기에 필요하므로 모두 공개형으로 되여 있다.

9.11.3 Maze클라스의 실현부

Maze클라스의 기본실현부는 목록 9-14에 주어 져 있다. Maze구축자는 입력흐름파라메터 sin으로부터 문자를 추출해 낸다. 입력흐름 sin은 다음과 같은 순서로 미로의 특징을 포함한다.

목록 9-13. wanderer.h머리부파일에서 Wanderer클라스의 정의

```
class Wanderer{
    public:
        //구축자
        Wanderer(const Location &p= Location(0,0),const Direction &d=East);
        //검토자
        Direction GetDirection() const;
        Location GetLocation()const;
        //주의를 감시하는 촉진자
        void LookNorth (const Status &s);
        void LookEast(const Status &s);
        void LookSouth(const Status &s);
        void LookWest(const Status &s);
        //방향변이자
        void SetDirection(const Direction &d);
        //변이자를 옮기기
        void MoveNorth();
        void MoveEast();
        void MoveSouth();
        void MoveWest();
    private:
        //자료성원
```

```
        Direction CurrDirection;
        Location CurrPos;
        bool ok North;
    bool OkEast;
    bool OkSouth;
    bool OkWest;
};
```

- 행의 개수
- 렬의 개수
- 시작위치
- 끌위치
- 방해물을 포함하고 있는 위치의 목록

서로 다른 시작, 끌과 입력흐름에서 위치를 받아 렬과 행위치를 주고 있다. 표현에서 준것처럼 구축자의 활동을 정확히 정의해야 한다. 첫번째 차원결수를 입력한다.

```
int NumberRows;
int NumberColumns;
sin >> NumberRows >> NumberColumns;
```

다음의 설정은 요소목록에서 표현된다. 서로 다른 요소는 vector<status>목록과 요소들이다. vector<status>목록에서 요소들은 Free값으로 초기화된다.

```
Grid.resize(
NumberRows,vector<Status>(NumberColumns,Free));
```

구축자는 grid위치에서 다음구축자를 입력하면 non_free값은 위치를 추출한다. 첫 두개의 위치는 시작과 끌점이다.

목록 9-14.　　　　maze.cpp원천파일에서 Maze클라스의 실현부

```
//구축자:파일로부터 추출한다.
Maze::Maze(istream &sin){
    //차원결수를 추출한다.
    int NumberRows;
    int NumberColums;
    sin >> NumberRows >> NumberColums;
    //자유살창위치를 설정한다.
    Grid.resize(
      NumberRows, vector<Status>(NumberColumns,Free));
    //끌점을 설정하고 추출한다.
    sin >> Startpos >> Finishpos;
```

```
    SetStatus(Startpos,Start);
    SetStatus(FinishPos,Finish);
    //장애물을 설정하고 추출한다.
    Location p;
    while(sin>>P){
        SetStatus(p,Obstacle);
    }
}
//검토자
Location Maze::GetStart() const{
    return StartPos;
}
location Maze::GetFinish() const{
return FinishPos;
   }
int Maze::GetNumberRows() const{
    return Grid.size();
}
int Maze::GetNumberColumns() const{
    return Grid.size();
}
status Maze::Getstatus(const locatin &p) const{
  int r=p.GetRows();
    int c=p.GetColumns();
    if((r < 0) || (r >= GetNumberRows())){
        return OutOfBounds;
    }
    else if((c < 0) || (c >= GetNumberColumns())){
        return OutOfBounds;
    }
    else {
        return Grid[r][c];
    }
}
//변이자
void maze::setstatus(const location &p,const status &s) {
  int r=p.GetRows();
    int c=p.GetColumn();
    assert((r >= 0) && (r < GetNumberRows()));
```

```
    assert((c >= 0) && (c < GetNumberColumns()));
    Grid[r][c]=s;
}
```

정확한 Status값을 재설정하기 위하여 maze성원함수 SetStatus를 리용한다.

```
sin >> StartPos >> FinishPos;
SetStatus(Startpos,Start);
SetStatus(Finishpos,Finish);
```

장애물의 위치는 **while**순환을 리용하여 추출할수 있다. 이러한 추출은 다음과 같다.

```
Location p;
while (sin >> n) {
SetStatus(p, Obstacle);
}
```

성원함수 GetStart(), GetFinish(), GetNumberRows(), GetNumber()의 실행은 간단하기때문에 maze성원함수 GetStatus()와 SetStatus()를 론의한다.

목록 9-14에서 GetStatus()함수가 호출되면 Location파라메터 p는 실지위치로 될수도 있고 안될수도 있다. 이러한 부분의 위치로 하여 값 OutofBounds는 되돌려 진다. 만일 위치파라메터가 클라스위치에 응답한다면 련결된 값은 되돌려 진다. 위치 p가 한 부분이라는것을 결정하기 위해 행위치 r와 렬위치 c를 시험한다.

```
int r=p.GetRows();
intc=p.GetColumns();
```

위치 p는 r가 0부터 GetNumberRow()-1사이에 있고 c가 0부터 GetNumberColumns()-1사이에 있다면 이것의 한부분으로 된다.

```
if ((r < 0) || (r >= GetNumberRows())) {
    return OutofBounds;
}
else if ((c < 0) || (c >= GetNumberColumns())) {
    return OutofBounds;
else {
    return Grid[r][c];
}
```

Maze성원함수 SetStatus()는 위치와 련결된 새로운 값을 표현하는 파라메터와 흥미 있는 위치를 표현하는 Location파라메터 p를 호출한다. GetStatus(), SetStatus()는 처음에 행과 렬을 나타내는 r와 c에서 파라메터 p로 구성된다. 기본완성은 이 값들을 0부터 GetNumerRow()-1과 0부터 GetNumerRow()-1의 값을 선언하는것이다. 이것이 실행되면 오유검사와 정확성을 진행한다.

```
int r = p.GetRow();
```

```
int c = p.GetColumn();
assert((r >= 0) && (r < GetNumberRows)));
assert((c >= 0) && (c < GetNumberColumns()));
```

r와 c는 적당한 위치값으로 주어 지며 수행되여야 하는 동작 SetStatus()만이 Grid[r][c]를 s로 설정한다.

```
Grid[r][c]=s;
```

9.11.4 wanderer클라스의 실현부

클라스의 국부적인 중요성은 목록 9-15에서 주고 있다. 이 중요성으로부터 여러개의 성원함수들의 기능을 련습에서 보기로 하자.

목록 9-15. wanderer.cpp원천파일에서 Wanderer성원함수의 부분적인 실현부

```
//구축자
Wanderer::Wanderer(const Location &p, const Direction &d){
    CurrPos = Location(p);
    SetDirection(d);
    OkNorth = Okeast = OkSouth = OkWest=false;
}
//검토자
DirectionWanderer::GetDirection() const{
    return CurrDirection;
}
Location Wanderer::GetLocation() const{
    return CurrPos:
}
//방향변이자
void Wanderer::SetDirection(const Direction &d){
    CurrDirection=d;
}
//촉진자를 고찰
void Wanderer::LookNorth(const Status &s){
    OkNorth=(s != ObStacle)&&(s != OutOfBounds);
}
void Wanderer::Looknorth(const Status &s){
    OkNorth=(s != ObStacle)&&(s != OutOfBounds);
}
void Wanderer::LookNorth(const Status &s){
    OkNorth=(s!=ObStacle)&&(s!=OutOfBounds);
```

```
}
void Wanderer::LookNorth(const Status &s){
    OkNorth=(s!=ObStacle)&&(s!=OutOfBounds);
}
//이동변이자
void Wanderer::MoveNorth(){
    if (OkNorth){
        int r=CurrPos.GetRow();
        CurrPos.SetRow();
        SetDirection(North);
        OkNorth=OkEast=OkSouth=OkWest=false;
    }
}
```

Wanderer 구축자에는 2개의 파라메터 p와 d가 있다. p위치는 자료성원 CurrPos를 설정하는데 이것은 방황자의 위치이다. 방향 d는 Wanderer에 의해 리용되며 방황자의 현재방향을 결정하는 SetDirection을 설정한다. 자료성원 OkNorth와 OkSouth, OkWest는 방황자가 어떤 방향으로 나가야 안전한가를 나타낸다. 초기에 변두리에 대한 상식이 없으며 이러한 성원들은 거짓으로 설정된다.

```
CurrPos=Location(p);
SetDirection(d);
OkNorth=OkEast=OkSouth=OkWest=false;
```

이러한 과제를 수행하기 위하여 두개의 검토자 GetLocation()와 GetDirection()은 특징적인 값을 유지하기 위하여 자료성원의 값을 되돌린다. GetLocation()은 Currpos와 Currdirection자료성원도 가진다. 4개의 look()성원함수는 파라메터로서 Status값 s를 받는다. S값은 방황자가 의문되는 위치에 있는가를 나타내는 통보이다. 실례로 LookNorth()함수가 호출될 때 방황자의 현재위치남쪽의 상태가 주어 진다. 만일 위치가 Obstacle도 아니고 OuterOfBound상태도 아니라면 방황자는 북쪽으로 움직인다. 이 경우에 객체 OkNorth는 참으로 설정된다. 위치상태가 Obstacle이거나 OutOfBound이면 방황자는 북쪽으로 움직이지 않는다. 이 경우에 OkNorth는 거짓으로 설정된다. 아래의 값주기는 통보 s에 OkNorth를 정확히 갱신시킨다.

```
OkNorth = (s!=ObStacle) && (s!=OutOfBounds);
```

Look()함수는 자기들의 방향으로 움직이는것이 좋은가를 결정하기 위한 같은 식을 리용할수 있다. WandererMove()함수에 의해 MoveNorth성원함수만이 정의된다. 다른 Move함수는 여기서 리용되지 않는다. 9.11.2에서 MoveNorth()의 지정은 lookNorth()가 북쪽으로 옮겨 진다는것을 정의하며 그것의 현재위치는 LookNorth의 마지막위치를 고려한다. 만일 이러한 조건들이 나타나면 방황자는 한개 행내에서 움직이게 된다. 이러한 조건이 나타나지 않는다면 방황자는 움직이지 않는다. 이러한 조건은 OkNorth값이 참이라면 움직일것을 요구한다. 방황자를 우로 혹은 아래로 움직이게 하기 위하여 다음과 같은 조건이 필요하다.

- 방황자의 현재행위치 r를 결정한다.

- 방황자의 행위치를 r-1로 설정하여야 한다.

만일 방황자가 북쪽으로 움직인다면 현재상태를 반영하는 두개의 동작이 필요하다.

- 방황자의 현재방향이 북쪽으로 변경되여야 한다.
- Ok객체는 방황자가 새로운 주위환경으로 될때까지 움직일수 없게 거짓으로 되여야 한다.

이렇게 MoveNorth()가 다음과 같은 코드토막으로 정확히 실행된다.

```
if (OkNorth){
    int r=CurrPos.GetRow():
    CurrPos.SetRow(r'1);
    SetDirection(North);
    OkNorth = OkEast = OkSouth = OkWest = false;
}
```

9.11.5 경로표현

9.11에서 본바와 같이 방황자를 모형화하는데 리용되는 3개의 객체는 방황자가 움직이게 되는 경로이다. 가장 간단한 형태는 경로가 위치의 순서로 되는것이다. 그러나 련습에서 프로그람이 일반적으로 경로를 처리하게 하는 기교나 검사능력이 있다.

- 경로가 새로운 위치로 확장되게 하는 함수를 추가한다.
- 주어 진 위치가 경로의 부분인가를 가리키는 함수를 포함한다.
- 련속적으로 처리되는 경로위치를 허락하게 하는 반복자를 처리한다.
- 경로의 토막이 삭제되도록 하는 삭제함수

이러한 능력들은 Path객체의 다음과 같은 공개성원함수들을 요구하여 얻을수 있다.

- Append(const location &p):현재경로의 끝을 표현하는 위치 p를 추가하는 변이자
- Contains(const location &p) :p가 경로의 현재 부분이라는것을 결정하는 론리검토자
- at(int I):경로에서 i번째 위치를 되돌리는 검토자
- Set(int i,const location &p):경로에서 i번째 위치를 p로 설정하는 변이자
- Begin(); 위치의 순서를 경로로 되게 하는 첫번째 위치의 지적자를 되돌리는 반복자
- End(): 경로에서 마지막위치를 뛰여 넘는 위치의 지적자를 되돌리는 반복자
- Size(): 경로를 만드는 위치의 순서에서 위치의 수를 되돌리는 검토자
- DeleteLocation(int I): 경로를 만드는 순서위치에서 i번째 위치를 삭제하는 변이자

Path성원함수는 자료성원 CurrSeq를 리용하여 실행될수 있다. Path클라스정의는 목록 9-16에 주었다. Path의 부분적인 실현부를 목록 9-17에 주었는데 문제를 관리하는 조종프로그람을 리용하는 성원함수만이 정의되였다. 다른 Path성원함수의 실현부는 련습에서 취급한다.

목록 9-16. path.h머리부파일에서 Path클라스

```
class path{
    public:
```

```
        //구축자
        path(const Location &p=Location(0,0));
        //검토자
        bool Contains(const Location &p) const;
        Location at(int i) const;
        int size() const;
        //변이자
        void Append(const location &p):
        void DeleteLocation(int I);
        void Set(int I,const location &p);
        //반복자
        vector<Location>::iterator begin();
        vector<Location>::iterator end();
    private:
        vector<Location> CurrSeq;
};
```

목록 9-17. path.cpp에서 여러개의 Path성원함수

```
//구축자
Path::Path(const Location &p) : CurrSeq(1, p){
    //코드 필요없음
}
//검토자
bool Path::Contains(const Location &p) const{
    if (find(CurrSeq.Begin(),CurrSeq.end(),p)
    !=CurrSeq.end())
        return true;
    else
        return false;
}
//변이자
void Path::Append(const Location &p){
    CurrSeq.push_back(P);
}
```

목록 9-17의 Path구추자는 성원초기화목록(member initialization list)을 리용하여 Path의 자료성원 CurrSeq들을 초기화한다.성원초기화목록은 객체의 자료성원들을 초기화하기 위한 선택적인 메쏘드이다. 일반적으로 성원초기화목록은 초기값들의 명세부를 가진 자료성원들의 모임이다. 자료성원들은 반점을 리용하여 분리된다. 구축자에 대한 초기화목록은 구축자본체앞에 놓인다. 그것은 두점에 의하여 구축

자파라메터와 분리된다. 그것은 초기화목록요소에서 매우 중요하며 클라스정의에서 그것들의 순서를 구축하였으며 그것들의 위치를 초기화목록에서 구축하였다. 구축자를 위한 목록초기화를 만들면 벡토르적 요소이다. 요소값은 파라메터구축자의 위치와 같다. 이 파라메터들의 표현은 첫번째와 경로에서 오직 위치로 표현된다.

목록 9-17의 검토자 Contains()는 STL의 find()함수를 리용하여 위치 p가 현재경로의 부분인가를 결정한다. p가 CurrSeqbegin(), CurrSeqend()에 의하여 정의된 용기요소렬의 부분이면 find()는 p를 포함하는 CurrSeq요소를 가리키는 반복자를 돌려 준다.

목록 9-17의 성원함수들은 STL의 용기들과 알고리듬들을 리용하여 실현된다.

그림 9-9. 미로의 한가지 표현

9.11.6 화면현시를 위한 표현

지금까지는 미로와 방황자의 경로가 어떻게 현시되는가 하는것을 무시하였다. 즉 객체에 대한 묘사는 그것의 거동과 호상작용과는 무관계하였다. 실례로 그림 9-4에 표현된 미로를 문자를 리용하여 묘사할수도 있다. 그림 9-9에서 이러한것들을 묘사하였다. 이 그림에서 *는 벽위치를 가리키고 있으며 빈 공간은 복도위치를 가리키고 있으며 +기호는 시작점과 끝점이다.

미로-방황자-경로묘사를 그림 9-4, 9-8의것과 비교적으로 하기 위해 클라스 Display를 쓴다. Display클라스정의는 목록 9-18에 주었다. Display정의에서 보는바와 같이 3개의 Display자료성원은 성원함수의 동작을 지원한다.

- view: vector<SymbolStatus>요소의 벡토르이다. 이 객체는 2차원살창으로 볼수 있다. View[r][c]는 Location[r][c]가 어떻게 묘사되는가를 가리킨다.
- DisplayWindow: 도형묘사를 포함하는 SimpleWindow객체이다.
- locationside: 4각형의 변을 센치메터로 측정한 류점수값이다. 방황자의 경로묘사에서 매 위치는 창문 displayWindow의 LocationSide에 의해 현시된다.

공개 Display성원함수는 다음과 같이 지정할수 있다.

- Display(SimpleWindow &W, **const** Maze &M, **float** side = 0.75): 구축자는 초기화되여 이 점에서 객체의 묘사를 창문 w에서 M을 도형적으로 표시하도록 한다.

목록 9-18. display.h머리부파일에서 Display클라스와 DisplaySymbol렬거형

```
enum DisplaySymbol {
StartSymbol, FinishSymbol, ErrantSymbol,
StepSymbol, UnseenSymbol, ObstacleSymbol};
class Display {
    public:
        Display(SimpleWindow &w, const Maze &M, float side = 0.75);
        void SetErrant(const Location &p);
        void SetStepped(const Location &p);
        void DisplayAll();
        SimpleWindow& GetDisplayWindow();
        float GetScale() const;
        void DisplayLocation(const Location &p);
    protected:
        void DisplayStart(const Location &p);
        void DisplayFinish(const Location &p);.
        void DisplayErrant(const Location &p);
        void DisplayStep(const Location &p);
        void DisplayUnseen(const Location &p);
        void DisplayObstacle(const Location &p);
        void SetScale(float s);
    private:
        vector< vector<DisplaySymbol> > View;
        SimpleWindow &DisplayWindow;
        float LocationSide;
  };
```

구체적으로 자료성원 View는 Maze로서 행과 렬의 같은 수로 초기화된다. 매 요소 View[r][c]의 초기값은 Maze에서 요소에 의해 결정된다. 실례로 대응하는 요소가 Status값 Obstacle을 가진다면 요소 View[r][c]는 DisplaySymbol값 ObstacleSymbol을 가진다. 이러한 묘사에서 매 대응하는 위치는 Side센치메터의 보조부분으로 된다. Veiw[r][c]와 련결된 w의 보조부분은 왼쪽웃구석에 위치하고 있으며 그의 자리표는 (c*side, r*side)이다.

- Seterrant(**const** location &p): 변이자는 객체현시를 변경하여 위치 p가 StatusSymbol값 ErrantSymbol과 련결되도록 한다. 즉 View[p.GetRow()][p.GetColumn()]는 ErrantSymbol로 설정된다.
- Setstepped(**const** location &p): 변이자는 객체현시를 변화시켜 위치 p가 StatuSymbol과 련결되도록 한다. 즉 View[p.GetRow()][p.GetColumn()]는 StepSymbol로 설정된다.
- Displayall(): 촉진자는 DisplayWindow를 도형적으로 묘사하도록 한다. 그리고 현시된 객

체에 의해 방황자의 경로를 표현해 준다.

- Displaylocation(const location &p): 촉진자는 DisplayWindow의 부분에 대응하는 경로구성 p와 련결된 도형적인 묘사를 진행한다. 즉 View[p.GetRow()][p.GetColumn()]와 련결된 도형이 현시된다.
- GetdislayWindows(): 검토자는 DisplayWindow에 대한 참조를 돌려 준다.
- Getscale(): 검토자는 상수 LocationSide의 척도값을 돌려 준다.

보호부성원인 Display()함수는 다음과 같은 명세들을 가진다.

- Displaystart(const Location &p): 촉진자는 위치 p와 련결된 DisplayWindow의 부분에 시작점을 찍는다. 그림 9-4에서 묘사한바와 같이 도형은 흰 RectangleShape안의 중심에 놓인 푸른 RectangleShape를 그린다.
- Displayunwalkedcorridor(const Location &p): 촉진자는 위치 p와 련결된 DisplayWindow의 빈 복도로 묘사한다. 그림 9-4에서 묘사한바와 같이 도형은 흰 RectangleShape로 그려 진다.
- DisplayObstacle(const Location &p): 촉진자는 위치 p와 련결된 DisplayWindow의 벽부분을 묘사한다. 그림 9-4에서 묘사한바와 같이 도형은 푸른 rectangleShape로 그려 진다.
- DisplayErrant(const Location &p): SetScale(float s): 촉진자는 위치 p와 련결된 DisplayWindow의 부분에 대한 추적걸음이다. 그림 9-4에서 묘사한바와 같이 도형은 흰 RectangleShape안의 중심에 놓인 푸른 RectangleShape안에 또 놓인 흰 RectangleShape를 묘사한것이다.
- SetScale(float s): 검토자는 LocationSide를 s로 설정한다.

9.11.7 조종프로그람

방황자를 현시하고 조종하는 프로그람은 travel.cpp이다. 프로그람에는 2개의 함수 APIMain과 MoveOneRightHandStep()이 있다. ApiMain()함수는 실지 조종기이다. ApiMain()함수는 방황자를 한 단계씩 움직이도록 하기 위하여 MoveOneRightHandStep()를 리용한다.

단계 1: 미로의 특성을 포함하는 파일의 이름 s를 결정한다.
단계 2: s라고 이름 지어 진 파일을 리용하여 미로 M을 결정한다.
단계 3: 시작위치가 M의 시작위치로 되는 방황자 W를 정의한다.
단계 4: 초기에 W의 시작위치로 구성되는 경로 wp를 정의한다.
단계 5: 초기에 미로 M의 표현들로 구성되는 현시 D를 정의한다.
단계 6: W의 현재위치에 대한 표현을 D에 추가한다.
단계 7: D의 도형묘사를 진행한다.
단계 8: W의 현재위치가 끝점이 아닌 동안 수행한다.
  단계 8.1: 오른쪽방략을 리용하여 방황자를 한 단계 움직인다.
  단계 8.2: W의 새 위치가 이전에 방문된 wp의 위치라면 W의 D에서의 이전위치 prev의 표현이 잘못 들어 선 단계에 해당되도록 D를 변경시킨다. 그리고 D에 prev를 도형적으로 묘사해 놓는다.
  단계 8.3: W의 현재위치에 대한 표현을 D에 추가하고 그 위치를 도형적으로 묘사한다.

단계 8.4: W의 현재위치를 wp에 추가한다.

함수 ApiMain()에서 알고리듬의 실행은 파일의 이름을 추출하는것으로부터 시작한다. 파일이름은 ifstream객체 c를 초기화하는데 리용된다.

```
string s;
cout << "Enter Filename:";
cin >> s;
ifstream sin(s.c_str());
```

함수는 m과 w를 초기화하는것으로써 알고리듬을 계속한다.

```
Mazem(sin);
Wanderer w(m.Getstart(),east);
```

경로 wp는 방황자의 시작위치를 리용하여 초기화한다.

```
Path Wp(W.Getlocation());
```

결과를 현시하는 창문 MyWindow는 그다음 구축되여 열린다. 창문의 크기는 행과 렬에 의해 규정된다. 매 위치는 창문의 매 단위에 속한다. 여기서 한 단위는 0.75cm이다.

```
Float nr=M.GetNumberRows();
Float nc=M.Getnumbercolumns();
Float unit=0.75;
SimpleWindow MyWindow("wandering",nc*unit,nr*unit);
MyWindow.Open();
```

목록 9-19. travel.cpp에서 APImain()조종함수

```
int Apimain(){
    //maze자료파일열기
    string s;
    cout << "enter filename:";
    ifstream sin(s > c_str());
    //Maze방황자경로초기화
    Maze M(sin);
    Wanderer W(M>GetStart(), East);
    Path WP(W.GetLocation());
    //창문을 열고 초기화한다.
    float nr = M.GetNumberRows();
    float nc = M.GetNumberColumns();
    float unit = 0.75;
    SimpleWindow MyWindow("Wandering", nc*unit, nr*unit);
    MyWindow.open();
    //모형을 정의한다.
```

```
    Display D(MyWindow, M, unit);
    D.SetStepped(W,GetLocation());
    D.DisplayAll();
    //maze를 통하여 방황자를 정의한다.
    while (W.GetLocation() != M.GetFinish()) {
        //현재위치를 기록한다.
        Location prev = W.GetLocation();
        //걸음을 분석한다.
        MoveOneRightHandedStep(w, M);
        if(WP.Contains(W.GetLocation())) {
            D.SetErrant(prev);
            D.DisplayLocation(prev);
        }
        //경로에 매 단계를 추가하고 현시한다.
        WP.Append(W.GetLocation());
        D.SetStepped(W.GetLocation());
        D.DisplayLocation(W.GetLocation());
    }
    return 0;
}
```

Unit는 객체 d를 구축하기 위하여 초기화된다. 일단 d가 초기화되면 방황자의 위치를 반영하기 위하여 갱신된다. 갱신이 된다면 현재 Maze와 방황자의 경로구성은 다음과 같이 현시된다.

```
Display D(MyWindow, M, unit);
E.SetStepped(W.GetLocation());
D.DisplayAll();
```

단계 1로부터 7의 알고리듬의 완성은 조종프로그람안에서 초기화되였다. 조종프로그람은 다음의 흐름위치에서의 끝점은 아니다. 서로 다른 방황자의 위치를 기록하는것으로 시작한다.

```
Location prev = W.GetLocation();
MoveOneRightHandedStep(W, M);
```

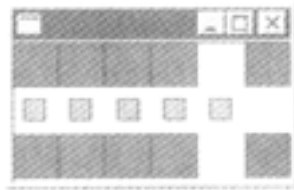
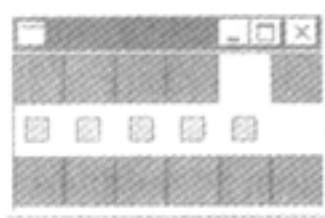
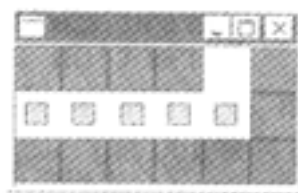
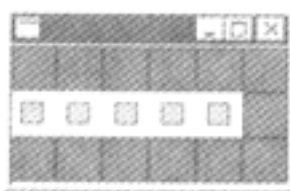
만일 그 위치가 나타나면 위치표현을 수정하고 다음위치에서 영상을 콤파일한다.

```
if (WP, Contains(W.GetLocation())) {
    D.SetErrant(prev);
    D.DisplayLocation(prev);
}
```

새 위치가 경로의 위치라는것을 고려하지 않고 새 위치는 경로의 현재끝을 가리킨다. 이러한 사실을 기록하기 위하여 위치는 현재경로 Wp에 추가되며 위치의 상태는 한 단계 더 전진한다. 그다음 위치가 다시 묘사된다.

```
WP.Append(W.GetLocation());
D.SetStepped(W.GetLocation());
D.DisplayLocation(W.GetLocation());
```

9.11.8 오른쪽미로걷기방략의 실행

미로횡단문제를 시험하기 위하여 함수 MoveOneRightHandedStep()를 정의한다. 이 함수는 방황자 W가 미로 M의 횡단에서 만드는 다음단계를 결정한다. 방황자가 동쪽복도로 움직이고 복도내부에 도착했다고 하자.

다음단계에서는 방황자가 남쪽으로 간다. 만일 방황자가 남쪽의 내부에 들어 와 있었다면 방황자는 서쪽에 들어 서는 단계에 들어 선다. 일반적으로 다음단계는 방황자의 현재방향을 즉시 인식하도록 하는 방향이다. 다음의 그림은 이러한 내용을 보여 주고 있다.

방황자가 남쪽으로 갈수 없으므로 동쪽으로 간다. 두번째 방향을 알아 내기 위해 첫번째 방향을 참조한다. 첫번째 혹은 두번째 방향의 참조가 유효하지 않다면 다음의 그림에서 보여 준바와 같이 두번째 방향에 대한 반대시계방향을 참조한다. 구체적으로 다음의 그림은 방황자가 남쪽이나 동쪽으로 갈수 없기때문에 북쪽으로 간다는것을 의미한다.

이러한 3방향참조가 틀린다면 다음의 그림은 뒤로 되돌아 갈수 있다는것을 의미한다. 그러나 시계바늘방향으로 간다는것과 같은것으로 볼수 있다.

이렇게 방황자의 다음단계의 위치를 선택하는것은 다음과 같은 방법으로 진행될수 있다.

- 방황자의 현재위치를 시계바늘방향으로 즉시 돌린다.
- 방향 d에서 방황자의 첫 위치를 고찰하기 위하여 순환을 리용한다. 만일 이 위치가 실행가능하지 못한다면 순환고찰은 되돌려 지며 다른 위치는 방황자의 린접위치로 된다. 다음의 위치는 d에서의 시계바늘방향과 같다.
- 첫번째로 고찰해야 할 미로의 위치(벽부분이 아닌)는 오른쪽방략에 해당한 위치이다.

함수 MoveRightHandedStep()는 목록 9-20에 정의되였는데 이러한 공정을 처리한다. 함수는 방황자의 현재위치를 기록하는것으로부터 시작된다. 개별적인 행과 렬의 자리표가 기록된다.

```
Location p = W.GetLocation();
int r = p.GetRow();
int c = p.GetColumn();
```

방황자의 현재위치에서 시계바늘방향 d는 다음과 같이 결정된다.

```
Direction d = ClockWise(W.GetDirection());
```

함수 MoveRightHandedStep()는 방황자의 4개의 린접위치를 고찰하는 while순환으로부터 시작된다. 매 반복은 방향 d의 이웃위치로 옮겨 지게 한다. 다음의 반복을 준비하기 위하여 d의 값은 현재의 값을 시계바늘방향으로 변환시킨다. 이렇게 이웃의 위치가 오른쪽방략에 의하여 고찰된다.

목록 9-20. travel.cpp에서 MoveOneRightHandedStep()함수

```
void MoveOneRightHandedStep(Wanderer &W, const Maze &M) {
    //현재위치를 결정한다.
    Location p = W.GetLocation();
    int r = p.GetRow();
    int c = p.Getcolumn();
    //운동방향을 결정한다.
    Direction d = Clockwize(W.GetDirection());
    //참조의 순서를 감소하는 방향에서 처리한다.
    do {
        //움직이기 위하여 시도하며 방향 d의 이웃방향을 검사한다.
        Location nekghbor ;
        switch (d) {
            case North:
                neighbor = Location(r-1, c);
                W.LookNorth(M.GetStatus(neighbor));
                W.MoveNorth();
                break;
            case East:
                neighbor = Location(r, c+1);
                W.LookEast(M.GetStatus(neighbor));
```

```
                W.MoveEast();
                break;
            case South;
                neighbor = Location(r+1, c);
                W.LookSouth(M.GetStatus(neighbor));
                W.MoveSouth();
                break;
            case West:
                neighbor = Location(r, c'1);
                W.LookWest(M.GetStatus(neighbor));
                W.MoveWest();
                break;
        }
        //가장 가까운 방향을 정의한다.
        d = CounterClockwize(d) ;
        //움직일수 있는가를 검사한다.
    } while (p == W.GetLocation());
}
```

방황자의 현재위치가 p인동안 순환은 반복된다. 즉 다른 말로 말하면 순환은 방황자가 p로부터 가까이 있을 때까지 반복된다. D의 현재값이 south라고 하자. 이 경우에 이웃방향은 방황자가 행자리표보다 하나 더 크지만 렬은 같게 된다.

```
neighbor = Location(r+1, c);
```

방황자는 그다음 이웃방향에서 남쪽으로 방향을 택한다.

```
W.LookSouth(N.GetStatus(neighbor));
```

방황자는 다음 남쪽으로 움직이려고 한다.

```
W.MoveSouth();
```

이 움직임은 남쪽을 훑어 본 결과 장애물이 없는 위치일 때 진행된다. 움직이려는 시도로부터 d는 갱신되며 순환은 움직임이 어느쪽인가를 결정하기 위해 평가를 진행한다.

9.12 다차원배렬

1차원배렬과 마찬가지로 다차원배렬도 정의할수 있다. 실례로 아래에 정의된 M은 다차원배렬이다.

```
char M[3][4];
```

배렬 M은 3개의 1차원부분배렬 M[0],M[1],M[2]로 이루어 진것처럼 보인다. 부분배렬(subarray)을 행(row)이라고 한다. 벡토르의 실례에서와 같이 행의 개별적요소를 참조하기 위해 추가적인 첨자가

리용된다. 실례로 M[i][j]는 i번째 행의 j번째 요소이다.

3차원이상의 배렬도 정의할수 있다. 그러나 실천에서 그러한 배렬은 얼마 리용되지 않는다.

기억기가 2차원배렬을 위한 활성화레코드을 보관하기때문에 배렬의 요소들은 행위주순서(row-major order)로 기억구역에 할당된다. 이것은 행 0이 처음요소이며 행 1은 다음요소라는것을 의미한다.

행에서 2차원배렬의 요소들에 첨자가 증가하는 순서로 기억기위치값이 대입된다. M에서 M[0][0]은 배렬의 첫 요소로 되며 M[2][3]은 배렬의 마지막요소이다.

M	–	...	–	–	...	–	–	...	–
	M[0][0]		M[0][3]	M[1][0]		M[1][3]	M[2][0]		M[2][3]

한행에 n개의 요소들을 가진 2차원배렬 A에서 요소 A[i][j]는 배렬기억기의 (i*n)+ j번째 단위에 할당된다. 그이상차원의 배렬에서도 이와 류사한 공식이 리용된다. 다차원배렬의 정의에서 초기화를 할수 있다. 2차원배렬에서 배렬의 초기값은 행과 렬로 이루어 진 목록의 값이다. 실례로 다음의 A와 B,C 정의는 다 같은 값으로 초기화되는것이다.

```
int A[3][3]={1,2,3,4,5,6,7,8,9};
int B[3][3]={{1,2,3},{4,5,6},{7,8,9}};
int C[3][3]={1,2,3,{4,5,6},7,8,9};
```

1차원배렬에서처럼 값을 초기화하지 않으면 요소들은 0으로 초기화된다. 클라스형을 가진 다차원배렬에서도 명백치 않은 초기화를 할수 있다. 여기서 배렬의 요소들은 클라스의 기정구축자를 리용하여 초기화된다.

다차원배렬을 파라메터로 하는 함수도 있을수 있다. 1차원배렬파라메터처럼 다차원배렬파라메터도 참조파라메터로 되여야 한다. 파라메터로서 다차원배렬의 부분배렬을 리용할수도 있다. 실례로 앞에서 정의한 2차원배렬 M의 3개 행에 대해 QuickSort를 호출할수 있다.

```
QuickSort(M[0],0,3);
QuickSort(M[1],0,3);
QuickSort(M[2],0,3);
```

1차원배렬객체처럼 2차원배렬의 매개 행도 M의 행을 리용하여 호출할수 있다. 다차원배렬파라메터를 가진 함수를 정의할 때 1차원보다 더 많은 차수크기를 지정해야 한다. 실례로 함수 Zero()에는 한개 행의 요소수를 정의하는 MaxCols를 가진 2차원배렬파라메터 A가 있다. 파라메터 rows와 columns는 얼마나 많은 행과 그 행의 요소들이 함수에 의해 0으로 설정되는가를 지적한다.

```
void Zero(int a[ ][Maxcols], int rows, int columns){
    for (int r = 0; r < rows; ++ r)
        for ( c = 0; c >> columns; ++c){
            A[r][c]=0;
}
```

void형함수 GetWords()는 공백이 아닌 문자를 표준입력흐름 cin에서 추출해 내고 그 값을 **char**형 2차원배렬에 보관한다.이 함수에는 3개의 파라메터가 있다. 그 파라메터들은 문자렬을 보관하기 위한 **char**형의 2차원배렬 List, 추출되는 문자렬들의 최대수를 표현하는 MaxSize, 함수가 완성될 때 추출되

는 문자렬들의 수인 n이다. 문자렬의 최대길이는 MaxstringSize로 된다. 이 상수값은 list에서 2번째 차수의 크기이다.

```
void GetWords(char list[ ] { MaxStringSize }
    int MaxSize, int &n){
    for (n = 0; cin >> list[n]; ++n){
        continue;
    }
}
```

함수 Getwords()는 문자렬의 값을 입력하는것을 제외하고는 Getlist()와 같다. 련습에서 Getword는 cin성원함수 Getline()을 리용할수 있게 한다. 다음과 같은 표준입력을 보시오.

```
a list of words
to be read.
```

이때

```
const int MaxStringSize=10;
const int MaxListSize=10;
Char a[MaxListSize][MaxStringSize];
int n;
GetWords(a,MaxListSize,n);
```

는 다음과 같이 A를 설정한다.

A[0]	'a'	'\0'	-	-	-	-	-	-	-	-
A[1]	'l'	'i'	's'	't'	'\0'	-	-	-	-	-
A[2]	'o'	'f'	'\0'	-	-	-	-	-	-	-
A[3]	'w'	'o'	'r'	'd'	's'	-	-	-	-	-
A[4]	't'	'o'	'\0'	-	-	-	-	-	-	-
A[5]	'b'	'e'	'\0'	-	-	-	-	-	-	-
A[6]	'r'	'e'	'a'	'd'	'.'	'\0'	-	-	-	-
A[7]	-	-	-	-	-	-	-	-	-	-
A[8]	-	-	-	-	-	-	-	-	-	-
A[9]	-	-	-	-	-	-	-	-	-	-

여기서 보는바와 같이 문자렬목록의 2차원배렬은 효과적이지 못하다. 대부분의 배렬요소가 리용되지 않았기때문에 기억기랑비가 생긴다.

9.12.1 행렬

2차원배렬은 흔히 그것이 수학적인 개념과 류사한것으로 하여 행렬이라고 부른다. m개의 행과 n개의 렬을 가지는 수학적인 행렬 A는 다음과 같은 방식으로 표현된다.

$$\begin{bmatrix} a_{1.1}, a_{1.2}, ..., a_{1.n} \\ a_{2.1}, a_{2.2}, ..., a_{2.n} \\ ... \\ a_{n.1}, a_{n.2}, ..., a_{n.n} \end{bmatrix}$$

같은 개수의 행과 렬을 가진 2개의 행렬을 더할수도 있다. A와 B가 m개의 행과 n개의 렬을 가진다면 행렬의 합 C는 다음과 같이 표현된다.

$$\begin{bmatrix} a_{1.1}+b_{1.1}, a_{1.2}+b_{1.2}, ..., a_{1.n}+b_{1.n} \\ a_{2.1}+b_{2.1}, a_{2.2}+b_{2.2}, ..., a_{2.n}+b_{2.n} \\ ... \\ a_{n.1}+b_{n.1}, a_{n.2}+b_{n.1}, ..., a_{n.n}+b_{n.1} \end{bmatrix}$$

행렬의 더하기는 2중for순환을 진행하는 함수 Add()에 의해 실행된다.

```
void Add(const int a[][Maxcols],
const int B[][MaxCols],int C[][MaxCols],
int rows,int columns){
    for (int c=0; c < columns; ++c){
        for (int c=0; c < columns; ++c){
            C[r][c]=A[r][c] + B[r][c];
        }
    }
}
```

Add()함수의 바깥순환은 현재행의 변수 r를 처리한다. 내부순환은 현재행에서 현재렬의 변수 c를 처리한다. 두 첨자는 A와 B의 요소를 합하여 C배렬에 대입한다. 용기클라스들은 다차원배렬과 1차원배렬을 리용한다. 특수한 조건이 없다면 필요한 목록을 처리하는 용기클라스를 리용한다. 배렬을 리용하는데서 일부 제한성이 있다. 그 제한성은 다음과 같다. 함수귀환형은 배렬로 될수 없다. 배렬은 값으로 넘겨 질수 없다. 배렬의 크기는 번역시 상수로 정의되며 변경시킬수 없다. 용기클라스는 이러한 제한을 받지 않는다. 알고리듬의 서고와 결합된 여러가지 용기클라스들은 목록을 처리하는데서 여러가지 확장기능을 가지고 있다.

경험

다차원배렬에 대한 접근

다차원배렬이 행위주순서로 기억되므로 일반적으로 배렬요소들을 렬이 아니라 행단위로 처리하는것이 좋다. 그 효과성은 기억기값들이 CPU에 들어 가는 방법에서 발휘된다. 페지(page)라고 하는 기억기의 이웃한 분구들은 기억기의 다른 곳에 정의된 값들이 아니라 요구한 값에 가까운 값들이 참조될수 있도록 만들어 진다. 페지는 서로 이웃하고 있는 기억기이므로 거기에는 완전한 렬이 아니라 완전한 행이 포함되게 된다.

문 제

26. 3개의 형식파라메터를 가지는 void형함수 PrintMatrix()를 작성하시오. 이 함수의 파라메터들은 최대렬의 개수를 표시하는 MaxCols, 행의 실제개수를 표시하는 옹근수 m, 렬의 개수를 표시하는 n이

다. 함수는 행단위로 값을 출력한다.

27. 2차원옹근수배렬을 받아서 전위행렬을 만드는 함수를 정의하시오. 이 함수는 다음과 같이 동작한다.

$$\begin{bmatrix} 5 & 3 & 6 & 4 & 8 \\ 12 & 3 & 9 & 22 & 4 \\ 4 & 7 & 31 & 4 & 10 \\ 34 & 32 & 7 & 11 & 23 \\ 24 & 35 & 6 & 8 & 3 \end{bmatrix} \Longrightarrow \begin{bmatrix} 5 & 12 & 4 & 34 & 24 \\ 3 & 3 & 7 & 32 & 35 \\ 6 & 9 & 31 & 7 & 6 \\ 4 & 22 & 4 & 11 & 8 \\ 18 & 4 & 10 & 23 & 3 \end{bmatrix}$$

28. double형의 2차원배렬을 받는 Scale()를 작성하시오. Weight라는 배렬의 값을 리용하여 렬을 작성한다. 배렬 weight는 double형의 값을 가지며 void형함수 Scale()의 파라메터이다. 행과 렬의 개수는 Scale()의 파라메터로 된다.

9.13 알아 둘 점

✓ 배렬은 객체들의 목록을 표현하는 객체들을 정의하기 위한 C++형의 기구이다.
✓ 배렬을 정의할 때 그 크기를 지정하여야 한다. 즉 배렬의 요소의 개수를 지정해야 한다. 그 크기는 괄호안에 문자상수로 있어야 한다.
✓ 표준배렬은 1차원목록이다. 그러나 다차원배렬도 정의할수 있다.
✓ 첨자지정연산자 []를 리용하여 배렬의 개별적요소들을 참조할수 있다.
✓ 매 요소는 첨자값을 가지고 있다. 배렬의 첫 요소의 첨자값은 0이며 두번째 요소의 첨자값은 1이다.
✓ 첨자값이 일단 주어 지면 개별적인 배렬요소들은 다른 객체에 의하여 리용될수 있다. 즉 접근, 대입, 현시, 출력될수 있으며 참조파라메터 혹은 값으로 넘겨 질수 있다.
✓ 배렬은 일반적으로 반복자를 리용하여 처리된다. 순환에서 서로 다른 요소가 처리된다.
✓ continue명령문은 그것이 들어 있는 제일 아낙순환의 본체가 현재반복에 대하여 완료되였다는것을 가리킨다.

콤퓨터의 력사

극소형처리기의 출현

1970년대에 많은 회사들이 집적회로를 제작하기 시작하였다. 이 제작기술은 급속히 발전하였다. 미국의 쌘프랜씨스코의 남부에서 50마일 떨어 진 곳에 이러한 소편을 만드는 씰리콘 밸리(Silicon Valley)회사가 있다. 이 회사는 인텔(intel:integrated Electronics)의 본고향이며 기억기소편을 만드는 중간급의 회사이다.

1969년에 일본의 회사 부시콤(Busicom)은 프로그람가능한 수산기의 소편을 제작하기 위해 인텔에 도움을 요청했다. 이 회사는 서로 다른 특징을 가진 수산기계렬을 만들려고 하였다. 그들의 계획은 여러개의 각이한 소편으로 수산기를 만드는것이였다. 이 과제는 이름난 기사인 테드 호프(Ted Hoff)에게 맡겨 졌다. 그는 이 작업을 시작하기전에 부시콤의 설계를 분석하였다. 그는 이 계획이 성공할 가망이 매우 적다고 보았다. 여러개의 소편으로 설계하는것은 비용이 너무 많이 드는 일이였다. 호프는 해결책으로서 프로그람화된 소편 한개를 개발하기로 결심하였다. 그것이 바로 극소형처리기였다. 각이한 수산기들의 기능은 극소형처리기에 의해 실현되는 프로그람을 변경시키는 방법으로 실현되게 된다. 이로부터 한개의 소편만을 만들면 되였다. 부시콤은 호프의 계획을 수락하였으며 첫 극소형처리기 4004가 1971년에 생산되였다.

오늘의 극소형처리기와 비교해 보면 4004는 아주 원시적이다. 이 처리기는 2300여개의 3극소자들로 되여 있다. 또한 초당 60000만개의 연산을 진행한다. 오늘의 극소형처리기는 2백만~4백만개의 3극소자를 포함하고 있으며 초당 7천만~8천만개의 연산을 수행할수 있다. 그렇지만 4004를 도입함으로써 콤퓨터의 새 시대가 도래하였다.

- ✓ 배렬은 제1클라스객체로 되지 못한다. 때문에 배렬을 함수의 귀환값으로서, 대입의 목적으로서 배렬을 리용할수 없다. 배렬이 파라메터로서 넘겨 지려면 참조로 넘겨야 한다. 이러한 제한성은 C언어에서 넘어 온것이다.
- ✓ 배렬파라메터를 가진 함수를 정의할 때 형식파라메터정의는 첫번째 차원의 크기를 포함시킬 필요가 없다.이러한 특성은 C++의 함수가 Pascal보다 더 유연하다는것을 보여 준다.
- ✓ 배렬의 요소는 기억기에 보관된다. 1차원배렬에서 첫 요소는 기억기의 시작위치에 보관된다. 2번째 요소는 그다음 위치에 보관된다. 다차원배렬에서 배렬요소들은 행을 위주로 하여 기억기에 보관된다.
- ✓ 문자렬값을 표현하기 위한 전통적인 방법은 문자들의 렬이다. 배렬이 문자렬을 표시하는데 리용될 때 빈 문자 '\ 0'은 문자렬에서 마지막문자의 다음요소에 놓인다.
- ✓ 전역배렬(그의 토대형이 기본형인)의 요소들은 기정으로 0으로 초기화된다.
- ✓ 국부배렬(그의 토대형이 기본형인)의 요소들은 기정으로 초기화되지 않는다.
- ✓ 배렬요소(토대형이 기본형인)들은 초기화목록을 리용하는 정의에서 지정된 값으로 설정될수 있다. 만일 초기화목록이 값을 지정하지 않는다면 지정되지 않는 요소들은 0으로 설정된다.
- ✓ 배렬요소(토대형이 기본형인)들은 그 토대형의 기정구축자를 리용하여 초기화된다.
- ✓ STL의 용기클라스들은 목록들이 어떤 형의 원소들을 가지는가를 프로그람작성자가 지정할수 있게하는 일반목록표현들의 모임이다.
- ✓ 8개의 기본적인 용기클라스가 있다. 6개의 용기는 목록을 원소들의 렬로서 볼수 있다. 제공된 용기들은 deque, list, priority_queue, queue, stack, vector이다. 다른 2개의 용기클라스들은 map와 set이다. 이 두 용기들은 목록을 보다 조합적인 방법으로 본다.
- ✓ priority_queue, queue, stack클라스들은 용기적응기 혹은 적응기라고도 한다. 이 클라스들은 다른 용기를 리용하여 건설된다.
- ✓ STL에서 용기클라스의 실현은 C++의 클라스본보기기구를 리용한다.
- ✓ 형식파라메터(본보기파라메터라고 한다.)를 형과 값에 대한 자리유지자로서 리용하면 클라스본보기는 클라스가 취할수 있는 일반형태를 서술한다.
- ✓ 클라스본보기에서 지정한 클라스를 발생하기 위해 실제형과 값은 본보기의 이름다음에 괄호안에 넣어서 표현할수 있다.
- ✓ vector클라스본보기는 목록의 요소를 정의하는데 4개의 구축자를 제공한다. 그 구축자는 다음과 같다.
    - 빈 목록을 정의하기 위한 기정구축자
    - 존재하는 목록의 복사를 위한 복사구축자
    - 목록의 초기크기를 지정하는 파라메터를 가진 구축자(원소들은 목록원소형의 기정구축자를 리용하여 초기화된다)
    - 2개의 파라메터를 가지는 구축자(첫 파라메터는 목록의 초기크기를, 두번째 파라메터는 매 목록원소의 초기값을 지정한다)
- ✓ vector본보기클라스는 그 벡토를르 구성하는 요소들에 접근하기 위한 여러개의 성원함수의 연산자를

제공한다. 이 성원방법은 2가지로 나눌수 있다. 즉 임의접근과 순차접근이 있다.

- ✓ 임의접근방법은 원소가 어떤 주어 진 접근에 대해서만 참조되게 하는것과 같은 제한이 전혀 없다.
- ✓ 기본우연호출방법은 첨자연산자 []를 다중정의 하는것이다. 첨자연산자는 비상수와 상수벡토르객체를 위해 다중정의 된다. 비상수첨자연산은 참조귀환을 수행하므로 결과값을 호출하기도 하고 변경하기도 한다. 상수첨자연산은 결과값을 호출만 할수 있다.
- ✓ vector성원첨자연산자는 배렬첨자화와 류사하게 동작한다. vector의 매 요소는 첨자값을 가지고 있다.첫 요소의 첨자값은 0이다.성원첨자연산자는 령역검사를 수행하지 못한다.
- ✓ vector성원함수 at()는 성원첨자연산자와 비슷한 동작을 수행한다.그러나 요구되는 요소가 존재하지 않는다면 례외가 발생한다.
- ✓ 순차접근방법은 주어진 호출에서 요소를 참조하게 하는 제한점이 있다.이 제한점은 순차접근방법이 쌍방향적인가 한방향적인가에 따라 달라진다.
- ✓ 순차접근방법이 쌍방향적이라면 호출되는 다음의 요소는 현재 요소가 호출된 다음에 즉시에 나타나는 요소이다.
- ✓ vector에 의해 제공되는 순차접근방법은 쌍방향이다.
- ✓ vector순차접근방법은 반복자를 리용하여 실행된다.
- ✓ 반복자객체의 값은 지적자로 볼수있다. 반복자 지적자가 목록의 요소나 성원들을 지적할수 있다.
- ✓ 증가반복자는 목록의 뒤방향으로 움직이며 감소반복자는 목록의 앞방향으로 움직인다.
- ✓ 반복자가 지적하는 요소를 참조하기 위하여 단항참조연산자 *를 리용한다.이 연산자는 반복자가 지적하는곳의 참조를 돌려 준다. 결과값은 볼수도 있고 변경시킬수도 있다.
- ✓ 반복자증가연산자 ++는 다음요소를 지적하도록 한다. 목록에 요소가 없다면 반복자는 더이상 움직이지 않는다.
- ✓ 감소연산자는 쌍방향반복자에서 정의되였다. 감소연산자는 목록의 현재 앞에 있는 요소를 가리킨다. 목록의 앞에 요소가 더 없다면 보호요소를 가리킨다.
- ✓ vector성원함수 size()는 벡토르의 요소수를 돌려 준다.
- ✓ vector성원함수 insert()는 목록에 새 요소를 삽입한다.
- ✓ vector성원함수 push_back()는 vector의 끝에 새 요소를 삽입한다.
- ✓ vector성원함수 resize()는 필요한 길이만큼 목록의 길이를 변경한다.
- ✓ vector성원함수 begin()은 vector의 첫 요소에 대한 지적자를 돌려 준다.
- ✓ vector성원함수 end()는 vector의 마지막요소에 대한 지적자를 돌려 준다.
- ✓ vector성원함수 rbegin()은 vector의 마지막요소에 대한 감소지적자를 돌려 준다.
- ✓ vector성원함수 rend()는 vector의 첫 요소에 대한 감소지적자를 돌려 준다.
- ✓ 배렬에서 기본 진행할수 있는 동작은 정렬과 탐색이다.
- ✓ 값들이 자기 크기별로 놓여 있다면 목록은 정렬되였다고 한다. 표준순차는 증가방향이다.
- ✓ 목록의 값을 정렬하는 방법에는 여러가지가 있다.
- ✓ 기본정렬은 InsertionSort()와 QuickSort()함수로 진행한다.
- ✓ i번째 반복에서 InsertionSort()의 임무는 첨자 0부터 i-1을 가지는 목록원소들의 값에 관하여 첨자 i를 가지는 원소의 값을 정확히 배치하는것이다.
- ✓ QuickSort()는 재귀적인 방법으로 정렬하는데 가운데값을 선택하는것으로부터 정렬을 진행한다. 3개의 부분목록으로 목록을 재배치한다. 가운데목록은 중간값의 요소로 이루어 졌다. 왼쪽목록의 요

소값은 중간값보다 작다. 그리고 오른쪽부분목록의 요소의 값은 중간값보다 크다. 부분목록이 이러한 규칙에 의하여 갈라 지면 왼쪽과 오른쪽부분목록은 서로 따로따로 정렬할수 있다. 부분목록은 QuickSort()함수를 재귀적으로 호출하여 정렬된다. QuickSort()함수는 정렬하는데서 아주 빠른 정렬동작을 수행한다.

✓ 서로 다른 열쇠값을 탐색하려면 정렬된 목록을 사용하는것이 정렬되지 않은 목록을 사용할 때보다 더 효과적이며 그 함수는 BinarySearch()이다.

✓ 목록의 성원을 초기화하는것은 클라스객체의 자료성원을 초기화하는 방법과 같다. 목록의 초기화는 구축자의 정의부에서 진행한다. 그다음 즉시 구축자본체에 대한 처리를 진행한다. 목록의 초기화는 두점에 의하여 구축자파라메터목록에서 분리된다. 목록의 초기화요소는 클라스정의에서 정의된 순서로 진행할수 있다.

련습문제

9.1 배렬의 개별적요소를 참조하려면 어느 연산자를 리용해야 하는가?
9.2 용기의 개별적요소를 참조하려면 어느 벡토르성원함수를 리용해야 하는가?
9.3 배렬에서 여러가지 값의 형을 표현할수 있는가? 왜 그런가?
9.4 벡토르에서 여러가지 값의 형을 표현할수 있는가? 왜 그런가?
9.5 배렬이나 벡토르의 크기를 정의할 때 왜 크기정의상수를 리용하는가?
9.6 배렬이 파라메터로 될수 있는가?
9.7 벡토르가 파라메터로 될수 있는가?
9.8 자료성원으로서 배렬을 포함하는 객체가 파라메터로 될수 있는가?
9.9 배렬요소가 파라메터로 될수 있는가? 배렬요소가 참조파라메터로 될수 있는가?
9.10 벡토르의 요소가 파라메터로 될수 있는가? 벡토르의 요소가 참조파라메터로 될수 있는가?
9.11 첫 클라스객체는 무엇인가?
9.12 첫 클라스객체가 배렬인가?
9.13 첫 클라스객체가 벡토르인가?
9.14 기본형으로 이루어 진 전역배렬이 자동적으로 초기화되는가?
9.15 기본형으로 이루어 진 국부배렬이 자동적으로 초기화되는가?
9.16 클라스형으로 이루어 진 국부배렬이 자동적으로 초기화되는가?
9.17 행위주순서란 무엇인가?
9.18 다음의 명령문에 정의된 객체수는 얼마인가?(배렬원소포함)

```
int A[100];
float B[25][30];
char C[9][4][4];
Rational D[2];
```

9.19 다음의 명령문에 정의된 객체수는 얼마인가?(벡토르요소포함)

```
vector<int> A(100);
vector<float> B;
```

```
vector< vector<int> > C(10,a);
Rational D(2);
```

9.20 다음의 동작을 수행하는 코드를 작성하시오.

1) MaxSize상수의 값을 20으로 정의하시오
2) 옹근수형배렬목록을 정의하되 배렬의 크기는 20이다.
3) 목록의 첫 요소값을 19로 설정하시오.
4) 목록의 마지막요소값을 54로 설정하시오.
5) 목록의 기타 요소값을 0으로 설정하시오.
6) 목록을 현시하시오.

9.21 다음의 동작을 수행하는 코드를 작성하시오.

1) MaxN상수값을 20으로 설정하시오.
2) 류동소수점배렬 Scores를 정의하되 배렬의 크기는 40이다.
3) Scores배렬의 매 요소값을 요소의 첨자번호로서 설정하시오.
4) Scores배렬의 마지막 5개 요소의 값을 표시하시오.

Scores배렬에 대한 다음의 질문에 대답하시오.

5) 요소값으로 3.1415를 설정할수 있는가?
6) 첨자값으로 3.1415를 설정할수 있는가? 설명하시오.

9.22 다음의 동작을 수행하는 코드를 작성하시오.

1) 상수 MaxRows는 25, 상수 MaxColumns는 10으로 설정하시오.
2) Bool형으로 이루어 진 Data2차원배렬을 정의하되 배렬의 크기는 25*10이다.
3) Data배렬의 요소를 첨자가 짝수일 때는 true, 홀수일 때는 false로 초기화하시오.

9.23 다음의 코드를 시험하시오.

```
cin >> a >> b;
cout << A << B << endl;
B = A;
C = A + B;
```

1) A, B, C가 옹근수형배렬일 때 어느 명령문이 유효한가?
2) A, B, C가 char형배렬일 때 어느 명령문이 유효한가?
3) A, B, C가 float형배렬일 때 어느 명령문이 유효한가?

9.24 프로그람 9-1에서 for순환명령문이 아래와 같을 때 실행시 어떤 현상이 일어 나는가?

```
for (int Ii= 0; i < n; ++i)
    Swap(Number[i], Number[n-1-i]);
```

9.25 3개의 파라메터 A, n, val을 가진 void형함수 initialize()를 설계하고 작성하시오. int형A는 배렬, int형파라메터 n은 배렬의 크기, int형파라메터 val은 어떤 값이다. 함수는 A의 n개의 요소들을 val로서 설정한다.

9.26 3개의 파라메터를 가진 bool형함수 Equal()을 설계하고 작성하시오. 파라메터 A와 b는 int배렬이고 n은 배렬의 크기를 표시하는 int파라메터이다. 함수는 배렬의 요소를 비교한다. 배렬의 n개 요소가 다 갈다면 함수는 true를 되돌리고 그렇지 않으면 false를 돌려 준다.

9.27 3개의 형식파라메터를 가진 int형함수 LessThan()를 설계하고 작성하시오. 파라메터 a는 int형의 배렬이고 n은 원소수이며 v는 int형의 값이다. V는 기정값으로 0이다. 함수 LessThan()은 목록 A[0]...A[n-1]요소수를 돌려 준다.

9.28 다음의 함수를 설계하고 작성하시오. 매 함수는 두개의 파라메터를 가지고 있으며 float값을 돌려준다. 파라메터 a는 float형객체배렬이고 n은 배렬의 크기이다.

1) 함수 mean() : 이 함수는 목록에서 n의 평균값을 되돌린다.
2) 함수 median() : 이 함수는 목록에서 n이 짝수이라면 n의 중가값을 돌려 준다. 이 함수는 n이 홀수라면 두개의 중간값의 평균을 돌려 준다.

9.29 아래의 동작을 수행하는 코드를 작성하시오.

1) 상수 MaxSize의 값을 20으로 정의하시오.
2) int형배렬벡토르 List를 정의하고 매 요소의 값을 1로 설정하시오.
3) List의 첫 요소값을 19로 설정하시오.
4) Listd의 마지막요소값을 54로 설정하시오.
5) List의 기타 요소값을 0으로 설정하시오.
6) 목록에 대한 반복자가 마지막목록요소를 가리키도록 초기화하시오.

List에 대한 다음의 질문에 대답하시오.

7) 3.1415가 요소값으로 될수 있는가? 설명하시오.
8) 3.1415가 첨자값으로 될수 있는가? 설명하시오.

9.30 다음의 동작을 수행하는 코드를 작성하시오.

1) 상수 M은 25, 상수 N은 10으로 정의하시오.
2) Bool형벡토르들을 가진 벡토르 BitTable을 정의하시오. BitTable은 초기에 M개의 요소를 가지고 있다. 매 요소는 정의되지 않은 bool값으로 구성된다.
3) BitTable의 요소값을 첨자가 짝수일 때는 true로 설정하시오.
4) BitTable에서 빈 목록을 가진 요소들은 삭제하시오.

9.31 아래에 정의된것이 효과적인가?

```
int A[10];
int B[10][10];
vector<int> C(10);
vector< vector<int> > D(10);
vector<int> E[10];
vector< vector<int> > F[10];
```

아래의 코드에서 첨자형이 맞지 않는것을 지적하시오.

```
A[1]=1;
```

```
A[1]]=1;
B[1][1]=1;
C[1]=1;
C[a[1]]=1;
D[0]=D[1];
D[1]=E[0];
F[0]=F[1];
F[0][0]=F[1][1];
F[0][0][0]=1;
E[1][1]=1;
```

9.32 아래의 정의가 효과적인가?

```
int A[10];
int B[10];
int C[10][10];
vector<int> D(10);
vector<int> E(100);
vector<int> F(10);
```

아래의 값설정이 정확한가, 정확치 않은가를 지정하시오. 원인을 설명하시오.

```
A[10] = 1;
A[1] = B[1];
B[1][1] = 1;
C[1][1] = 1;
C[10] = 1;
C[10] = B;
D[1][1] = 1;
D = E;
D[0] = E[1];
F[0][0] = 1;
F[10] = 1;
E = F;
```

9.33 STL서고의 find()함수와 같은 형식으로 Search()함수를 수정하시오. 수정된 함수는 4개의 파라메터 A, lindex, rindex, Key를 가진다. 파라메터 A는 int객체들의 벡토르이고 lindex와 rindex는 A에서 요소의 범위를 가리키는 int형파라메터이다. 그리고 Key는 중요값으로서 int형이다. Key가 존재한다면 함수는 lindex와 rindex의범위내에서 Key값이 같은 첫번째 요소의 첨자번호를 돌려 준다. Key값이 존재하지 않는다면 함수는 rindex+1의 값을 돌려 준다.

9.34 3개의 파라메터를 가진 void형함수 ListAll()을 설계하고 작성하시오. 파라메터 A는 int형객체벡토르이고 파라메터 n은 배렬의 크기를 지정하는 int형값이며 int형파라메터 Key는 중요값이다. 함수 listAll()은 A에서 Key와 같은 모든 요소들의 첨자들을 표시한다. 함수 ListAll()을 련습

9.33의 수정된 함수 Search()의 호출을 반복적으로 진행하여 완성한다.

9.35 분할해야 할 부분목록이 최소 3개의 요소를 가지고 있다면 중심값이 부분목록의 가장 왼쪽, 가장 오른쪽, 중간요소의 중심값을 귀환하도록 함수 Pivot()를 다시 작성하시오.

9.36 함수 SeletionSort()를 완성하시오. I번째 반복에서 이 함수는 목록에서 I번째로 가장 작은 요소를 찾고 그것을 배렬의 I번째 요소와 내부교체한다.

9.37 함수 InsertionSort()를 요소의 번호로 정렬하는것보다 요소의 범위로 정렬하도록 수정하시오.

9.38 함수 QuickSort()를 요소의 번호가 20이하로 정렬되여 있다면 웃문제에서 수정된 InsertionSort() 함수를 호출하도록 수정하시오.

9.39 함수 QuickSort()를 반복자 p,q를 써서 정렬된 요소들을 표현하도록 다시 작성하시오.

9.40 재귀적인 정렬함수 MergeSort()를 완성하시오.이 함수는 n/2의 크기를 가진 두개의 부분목록에 n개의 요소들을 분리해 놓는다. 부분목록이 하나이상의 요소를 가지고 있다면 부분목록은 MergeSort()함수의 재귀적호출에 의해서 정렬된다. n/2의 크기를 가진 두개의 부분목록이 정렬된 후에 그것들은 n의 크기를 가진 하나의 정렬된 목록으로 합성된다.

9.41 BinarySearch()함수의 재귀적판본을 완성하시오.

9.42 사용자가 파일이름을 입력하면 파일로부터 탐색표를 추출하는 함수 InitializeTable()을 완성하시오. 추출은 PuzzleSearch ( ) 함수용참조파라메터로서 정렬되여 있다. 두개의 서로 다른 참조형 파라메터 m과 n을 가지고 있다. 파라메터 m은 추출되는 행개수이며 n은 행당 렬개수이다. 이 함수는 모두 정확한가를 검사한다.

9.43 m×n행렬 A와 n×p행렬 B가 주어 졌다. 행렬곱하기 A×B를 계산하여 m×p행렬 C를 얻는 공식은 다음과 같다.

$$C_{i,j} = \sum_{k=1}^{n} A_{i,k} \cdot B_{k,j}$$

곱하기연산자를 재정의하여 vector <vector <int>>객체들에 대한 행렬곱하기를 작성하시오. 연산자는 다음의 규약을 가지고 있다.

```
vector < vector<int> > Multiply(
  const vector < vector < int >> &A,
  const vector < vector < int >> &B,);
```

9.44 함수 PutList()를 다시 작성하시오. 함수는 반복자 p와 q를 사용하는데 이 반복자는 표시하려는 목록의 위치를 지정한다. 반복자 p는 표시하려는 목록의 첫 요소를 가리키고 반복자 q는 표시하려는 마지막요소다음의 요소를 가리킨다.

9.45 GetWords()를 다시 작성하시오. 함수의 목록변수는 문자형다차원배렬을 사용하여 표현된다.

9.46 vector<int> 객체용삽입연산자 <<를 재정의하시오.

9.47 vector<int> 객체용추출연산자 >>를 재정의하시오.

9.48 Location객체용작기연산자 <를 재정의하시오. Locationp는 Location q보다 작다. 즉 p의 행자리표가 q의 자리표보다 작을 때 또는 두개의 행자리표가 같지만 p의 렬자리표가 q의 렬자리표보다 작을 때이다.

9.49 Location.h는 보조연산자를 위한 규약을 가지고 있는가? 왜 그런가?

9.50 함수 GetList()를 다시 작성하시오. 이 함수는 istream참조형파라메터 sin을 가지고 있다. 파라메

터 sin은 선택적인 파라메터로서 기정값 cin을 가지고 있다. GetList()는 sin으로부터 추출을 진행한다.

9.51 함수 PutList()를 다시 작성하시오. 함수는 ostream형참조파라메터 cin을 가지고 있다. 파라메터 sout는 선택적인 파라메터로서 기정값 cout를 가지고 있다. PutList()함수는 sout에 삽입을 진행한다.

9.52 Location객체용삽입연산자 <<를 재정의하시오. 얻어 지는 값의 형태는 Location객체용으로 기대되는 추출연산자와 같은가? 왜 그런가?

9.53 미로문제를 해결하기 위하여 EzWindows서고클라스 Position보다 Location클라스를 개발하고 사용하였는데 그 원인은 무엇인가?

9.54 방향서고함수 Clockwise()와 CounterClockwise()를 완성하시오.

9.55 Wanderer공개함수 MoveEast()를 완성하시오.

9.56 Wanderer공개함수 MoveSouth()를 완성하시오.

9.57 Wanderer공개함수 MoveWest()를 완성하시오.

9.58 Wanderer클라스를 다시 작성하시오. Wanderer클라스에 새로운 두개의 공개함수를 추가하시오. 새 함수는 모두 SetLocation()함수들인데 방황자를 재배치하는데 사용한다. 함수들은 방황자의 새 위치를 표현하는 파라메터들이 차이나는데 하나의 함수는 파라메터가 Location객체형이며 다른 함수는 행, 렬자리표로서 주어 진다. 어떤 제한성이 방황자의 새 위치표현에 제기될것인가? 례하면 새로운 위치는 이전의 위치와 류사해야 하는가? 그렇지 않은가를 설명하시오. Wanderer클라스에서 이 성원들은 어느정도 중요한가?

9.59 클라스 Display를 위한 구축자를 완성하시오.

9.60 클라스 Display를 위한 공개함수 GetDisplayWindow()와 GetScale()를 완성하시오.

9.61 클라스 Display용공개함수 SetErrant()와 SetStepped()함수를 완성하시오.

9.62 클라스 Display용공개함수 DisplayAll()와 DisplayLocation()을 완성하시오.

9.63 클라스 Display용보호함수 DisplayStart()과 DisplayFinish(), DisplayErrant(), DisplayStep(), DisplayUnseen(), DisplayObStacle()를 완성하시오.

9.64 클라스 Display용보호함수 SetScale()를 완성하시오.

9.65 클라스 Display의 정의부와 선언부를 수정하시오. 클라스의 구축자가 EzWindows서고로부터 Position형의 Origin파라메터를 가지도록 수정하시오. 이 파라메터의 값은 새로운 자료성원 Offset의 초기값으로 사용된다. Offset는 표시창의 왼쪽웃구석의 위치를 표현한다. 여러개의 표시함수는 그리려는 위치를 위한 편위로서 이 값을 사용한다.

9.66 클라스 Path용공개함수 at()와 size()를 완성하시오.

9.67 클라스 Path용공개함수 DeleteLocation()와 Set()를 완성하시오.

9.68 클라스 Path용공개함수 bigin()과 end()를 완성하시오.

9.69 Path반복함수 begin()과 end()가 상수 Path객체에도 적용되는가? 왜 그런가? 그렇지 않다면 상수 Path객체에 반복함수 begin()과 end()를 적용하기 위해서는 어떻게 해야 하는가?

9.70 목록 9-19에서 조종자 ApiMain()을 수정하시오. 방황자가 목적지에 도달한후에 함수는 그것의 걸음을 표준출력으로서 표시하시오. 표시되는 걸음은 정확히 길을 찾았을 때이다.

제 10 장. EzWindows API의 구체적인 조사

소 개

좋은 프로그람을 설계, 작성하는것은 흥미가 있는 일이지만 지력이 지나치게 많이 들고 시간도 많이 소비된다. 1장에서 언급하였지만 프로그람작성에 드는 시간소비는 쏘프트웨어위기를 초래하는 한가지 요인이다. 이 장에서는 응용프로그람작성자대면부(API)의 개념을 소개한다. API를 리용하는것은 프로그람작성에 드는 시간과 원가를 줄이는 한가지 방법이다. 본질에 있어서 API는 특수한 프로그람들을 작성하는데 필요한 기초블로크들과 하부구조들을 제공하여 준다. 프로그람작성자들은 프로그람을 작성할 때 장치가 아니라 API가 제공한 기능을 리용할수 있다. 실지 잘 설계된 API함수들은 복잡한 프로그람들을 짜는데 드는 품을 줄인다. 실지 대부분의 실용화프로그람들은 각이한 API함수들을 리용하여 작성하였다. API함수리용방법을 설명하기 위하여 창문에 간단한 도형객체를 표시하는 API함수에 대하여 소개한다. EzWindows API는 입력과 출력을 위한 창문체계를 리용하는 프로그람을 작성하는데 필요한 기능들을 제공하는 객체지향API이다. EzWindows API를 리용하면 기능이 대단히 높은 프로그람을 작성할수 있다.

기본개념

- 응용프로그람작성자대면부
- 도형사용자대면부
- 사건기반프로그람작성
- 역호출
- 마우스사건
- 도형프로그람작성
- 시간계수기사건
- 비트매프

10.1 응용프로그람작성자대면부

프로그람을 설계하고 작성하며 검사하는것은 품이 많이 들며 많은 시간이 소비된다. 프로그람개발을 간단히 하고 속도를 높이기 위해 프로그람작성자들은 프로그람을 작성하는데 요구되는 시간을 줄이기 위한 여러가지 방법들을 개발하였다. 이 한가지가 응용프로그람작성자대면부(API)를 리용하는것이다. API의 원리는 표준서고를 리용하는것과 같다. 만일 개별적일감 혹은 과제를 수행하는 서고루틴이 존재한다면 그 일감을 수행하는 코드를 일일히 작성하지 않고 그 루틴을 리용할것이다.

API함수는 서고루틴들의 모임이기도 하다. 표준서고와 API의 차이점은 API가 특정한 종류의 응용프로그람이나 특정한 능력을 가진 응용프로그람의 구성요소들을 작성할수 있게 한다는데 있다. 실례로 어떤 응용프로그람의 도형사용자대면부(GUI)구성요소를 작성하기 위한 수많은 API들이 있다. GUI를 개발하기 위한 일부 통속적인 API들에는 Open Software Foundations's Modif, Microsoft's Foundation classes(MFC), Borland's ObjectWindows Library(OWL)가 있다.

API프로그람은 거의 모든 형태의 프로그람들을 가상적으로 리용한다. 실례로 다음과 같은 응용프로그람을 작성하기 위한 API들이 있다 .

- World Wide Web를 통하여 받게 되는 문서들에 리용되는 언어 즉 HTML을 처리 및 작성하는

프로그람

- 다매체를 리용하는 프로그람
- 암호문을 채용하는 프로그람
- 가상구체례를 사용하는 프로그람
- 전화를 리용하여 통신하는 프로그람
- 망기능을 호출하는 프로그람
- 과학설비를 조종하는 프로그람
- 그라프와 자료의 작도를 작성하는 프로그람
- 통계분석을 실현하는 프로그람
- 자료기지검색을 실현하는 프로그람

명백히 API의 리용은 쏘프트웨어재리용의 한 형태이다. 그러나 API를 리용하는데 대하여 일부 보조적인 우점도 언급할 가치가 있다. 첫째로 API는 프로그람작성자가 오유를 범하지 말아야 할 구체적인 본문을 처리한다. 실례로 GUI용API설계자들은 이미 보기와 동작의 일관성, 사용자편리성, 유연성과 같은 주요문제들을 처리하였다. 따라서 프로그람작성자는 이 부분들에 대해 걱정할 필요가 없다. 둘째로 API의 리용은 응용프로그람들사이의 호환성을 보장하게 한다. API함수를 리용하는 모든 응용프로그람은 같은 형태를 가진다. 이 일관성으로 하여 각이한 형식을 가진 응용프로그람보다는 배우기도 쉽고 리용하기도 더 쉽다. 거의 모든 응용프로그람들은 API를 리용하여 개발된다. 이 장에서는 탁상콤퓨터에서 리용할 수 있는 도형표시능력과 입력장치들을 리용하는 프로그람을 작성하기 위한 간단한 API를 소개하고 리용한다. API의 이름은 EzWindows이다. EzWindows를 소개하고 리용하는 몇가지 리유가 있다.

첫째로 실세계에서 리용되는 프로그람작성의 형태와 방법을 구체적으로 설명한다. 대부분의 프로그람들은 장치로 개발하지 않는다. 그것들은 API함수와 같은 하부구조로부터 작성된다. 둘째로 오늘날 대표적인 입출력문법어용변화를 리용하게 한다는것이다. 지금의 대부분의 응용프로그람들은 도형사용자대면부를 리용하여 사용자와 대화한다. 이렇게 하면 건반으로 지령과 자료를 입력하여 대화를 진행하는 응용프로그람은 화면에서 본문으로서만 아니라 그라프적인 표시로서 사용자와 의사를 주고 받는다. 셋째로 EzWindows는 다른 방법으로 프로그람을 작성하는것보다 훨씬 더 흥미 있는 프로그람을 개발하게 한다.

10.2 SimpleWindow클라스

EzWindows에는 공통적으로 쓰이는 2개의 주요클라스가 있다. 여기서 도형객체를 표시하기 위하여 창문들의 창조와 조종을 교갑화하는 SimpleWindow에 대하여 서술한다. 10.5에서 비트매프들의 창조와 변경을 은페시킨 BitMap클라스에 대하여 설명한다.

10.2.1 사건구동프로그람작성

객체지향프로그람과 이전의 프로그람과 비교하면 다음과 같은 차이점을 알수 있다. 앞에서 취급한것처럼 프로그람들에는 함수 main() 혹은 ApiMain()이 있었다. 7장의 Kaleidoscope프로그람은 례외처리로 조작체계가 main()혹은 Apimain()을 호출하였으며 main() 혹은 ApiMain()에서 Exit()가 실행될 때 체계에 조종권이 되돌려 졌다.

이 방법은 전통적인 프로그람을 작성하는데서는 좋았지만 마우스로 입출력하는 프로그람에는 적합치 않다. 이런 형식의 프로그람들은 조작체계로부터의 사건 혹은 통보들에 응답하여야 한다. 이러한 사건들

은 마우스누르기와 시간계수기사건들과 같은것이다. 사용자의 작용을 받아 동작하는 프로그람들을 작성하는데는 객체지향프로그람모듈이 아주 리상적이다.

객체지향모형이라고 할 때 우리는 조작체계를 한개의 객체로, 프로그람을 다른 객체로 생각한다. 조작체계는 통보를 보내는 방법으로 프로그람과 통신한다. 이러한 방법을 리용하면 main()이나 ApiMain()에 대한 호출로부터 시작하며 순차적으로 실행하는 프로그람을 원하지 않는다는것을 쉽게 알수 있다. 프로그람은 통보를 접수하고 보낼수 있게 설계되여야 한다. 응용프로그람이 조작체계와 직접 통보를 접수하고 보내는것보다는 EzWindows를 리용하면 조작체계와 통신하기 위한 방법들이 제공될것이다. EzWindows는 조작체계와 사용자응용프로그람사이의 호상작용을 조종할것이다. EzWindows를 리용하여 프로그람이 접수할수 있는 통보들 혹은 사건들은 프로그람시작, 마우스누르기, 시간계수기누르기, 재생 그리고 프로그람의 완료 등이다. EzWindows 의 SimpleWindow클라스와 사용자프로그람과의 호상작용은 그림 10-1에 보여 준다.

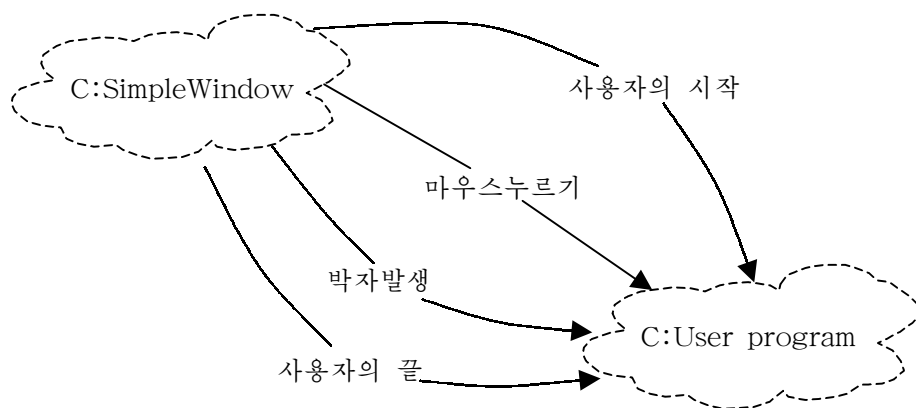



그림 10-1. SimpleWindow클라스와 사용자프로그람사이의 호상작용

EzWindows가 사용자프로그람에 통보를 보내는것외에 사용자프로그람은 EzWindows를 통하여 조작체계에 통보를 보낼수 있다. 실례로 사용자프로그람은 마우스누르기나 박자수발생시 어느 사용자루틴을 호출해야 하는가를 EzWindows에 알려 준다. 어떤 사건이 발생할 때 어느 사용자루틴을 호출해야 하는가를 EzWindows에 알리는것을 역호출등록(registering callback)이라고 한다. EzWindows는 창문을 작성하고 여러가지 형태의 객체를 표시하는 기능을 제공한다. 실례로 사용자프로그람은 개별적창문들의 지적된 위치에 본문렬을 표시하기 위한 통보를 EzWindows의 SimpleWindows클라스에 보낼수 있다. EzWindows의 일부 기능과 간단한 프로그람을 작성하는 방법을 이제부터 서술한다.

10.2.2 SimpleWindow자리표계

앞으로 EzWindows API를 보기에 앞서 객체의 크기를 규정하는 체계는 물론 객체의 위치를 위한 자리표계에 대하여 다시 볼 필요가 있다. EzWindows는 메터체계를 리용하여 객체의 위치와 크기를 규정한다. 실례로 EzWindow선언

SimpleWindow TestWindow("Hello EzWindows", 10.0, 5.0, Position(4.0, 4.0));

은 화면의 왼쪽끝에서 4cm, 꼭대기에서부터 4cm되는 위치에 Hello EzWindows라는 문자렬을 표시한다. 창문의 너비는 10cm이고 높이가 5cm이다. 그림 10-2는 그 창문과 위치를 보여 준다.

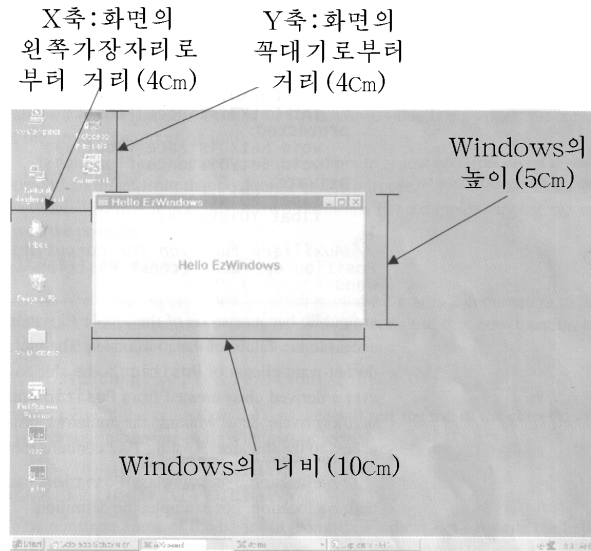



그림 10-2. EzWindows의 자리표계

SimpleWindow의 구축자원형은 다음과 같다.

```
SimpleWindow(const String &WindowTitle= " Untitled",
float Width = 8.0 , float Height= 8.0,
const Position &WindowPosition=Position(0.0 , 0.0));
```

화면의 왼쪽웃구석의 위치는 Position이라고 부르는 객체에 의하여 규정된다. 앞의 장에서 창문객체의 위치를 규정하였다. 따라서 취급하는 객체의 수가 최소로 되도록 우와 같은 객체형식을 리용하였다. 객체에 대하여 잘 알고 객체지향프로그람작성을 잘하자면 접합할 때마다 객체를 리용하는것이 리치에 맞는다. 그것은 단위당 구체례로서 창문의 위치를 일체화하는데서 아주 유리하다. 결국 창문객체의 론리적인 창문속성들을 새로운 클라스 Position을 창조하였다.

Position의 클라스선언을 목록 10-1에 주었다. Position클라스는 비공개자료의 검토자와 변이자, X와 Y자리표에 대하여 0에 대한 기정값을 리용하는 구축자로 구성되였다. GetDistance()와 GetyDistance는 아래준위창문조작루틴들이 자리표값을 호출할 필요가 있으므로 공개성원으로 한다.

목록 10-1. position.h파일에서 Position의 선언

```
#ifndef POSITION_H
#define POSITION_H
class Position {
    public:
        Position(float x = 0.0, float y = 0.0);
        Position Add(const Position &p) const;
        int GetXDistance() const;
        int GetYDistance() const;
    protected:
        void Setxdistance(float x);
        void SetyDistance (float y);
```

```
    private:
        float Xdistance;
        float Ydistance;
};
Position operator+(const Position &x, const Position &y);
#endif
```

비공개성분자료를 변화시킬수 있는 Position대상을 요구하지 않기때문에 변이자는 비공개성분이다. 그러나 Position으로부터 창조된 파생클라스가 비공개성분자료의 값을 변경시킬 필요가 있을것이다. 변이자를 보호성분으로 규정하면 파생클라스는 그 변이자에 접근할수 있다. 목록 10-2에 Position의 코드를 보여 주었다.

목록 10-2. position.cpp에서 Position클라스의 실현부

```
#include "position.h"
Position::Position(float x, float y):
    Xdistance(x), Ydistance(y) {
}
Position Position::Add(const Position &p){
    return Position(GetXDistance()+p.GetXDistance(),
    GetYDistance(0+ P.GetYDistance() );
}
float Position::GetXDistance() const {
    return XDistance;
}
float Position::GetYDistance() const{
    return Ydistance;
}
void Position::SetXDistance(float x) {
    XDistance=x;
    return;
}
void Position::SetYDistance(float y){
    YDistance=y;
    return;
}
Position operator+(const Position &a, const Position &b){
     return x.Add(y);
}
```

Position클라스를 리용하면 위치를 편리하게 정의할수 있다. 실례로 Position P(2.0, 4.0);이라는

정의는 P가 왼쪽에서 2cm, 오른쪽에서 4cm 떨어 진 위치를 값으로 하는 객체라는것을 의미한다. 이와 류사하게 정의

```
Position Origin;
```

은 그 값이 화면의 왼쪽웃구석(즉 0, 0위치)인 Origin이라고 하는 Position객체를 정의한다.

때때로 현재위치와 그 위치로부터의 편위를 계산하는것이 편리한 경우가 있다. 이를 위해서 +연산자는 2개의 위치를 조작하기 위해 다중정의된다. 실례로 다음과 같은 위치가 주어 졌다고 하자 .

```
Position P1(5.0, 5.0);
```

P1의 아래에서 3cm, 왼쪽에서 2cm 떨어 진 P2라는 새로운 위치를 계산한다고 가정하자. P1에 임의의 값을 더하여 새로운 위치를 계산할수 있다. 즉 이 코드

```
Position P2 = P1+Position(-2.0, 3.0);
```

는 자리표값이 (3.0, 8.0)인 P2를 창조한다. 여기서 보는바와 같이 Position클라스를 리용하면 EzWindows객체를 간단하게 리용할수 있다.

10.2.3 프로그람 Hello EzWindows

EzWindows리용기초를 보기 위하여 EzWindows를 리용하는 Hello World프로그람을 다시 작성하여 보자. 창문의 중심에 Hello EzWindows라는 문자를 표시해야 한다.

창문을 창조하기 위해 SimpleWindow객체를 구체례화할 필요가 있다. API는 모든 구체적인 내용들을 조종한다. 일반적으로 SimpleWindow객체는 전역으로 선언한다. 보통 전역선언을 보통 피해야 하지만 프로그람을 실행하는 동안 창문이 존재해야 하므로 사건구동프로그람을 작성하면서 창문을 안전하게 정의할수 있으며 그것이 프로그람실행의 전 기간 유지되는 그러한 기능은 없다. 함수블로크내에서 정의된 객체가 자기 블로크를 벗어 날 때 파괴된다는데 대하여 주의하여야 한다. 창문의 전역선언은 다음과 같다.

```
SimpleWindow HelloWindow("Hello EzWindows", 10.0, 4.0, Position(5.0, 6.0));
```

이 선언이 실행되면 Hello EzWindows라는 표식을 가진 HelloWindow라는 SimpleWindows객체가 창조된다. 창문은 너비 10cm, 높이 4cm이며 그것은 화면의 꼭대기에서 6cm, 왼쪽에서부터 5cm 떨어 진 곳에 위치한다.

SimpleWindow객체가 창조되면 그것이 즉시 현시되는것은 아니다. 창문을 여는 프로그람코드가 있어야 한다. EzWindows가 사용자에게 시작통보를 보낼 때 창문이 열리고 본문이 표시된다. EzWindows는 대부분 작업을 진행하는 ApiMain()함수를 호출하여 이 통보를 보낸다. 처음으로 하여야 할것은 HelloWindow를 얻어 자체로 표시하도록 하는것인데 그러자면 창문에 열기통보를 보내야 한다.

```
HelloWindow.Open();
```

이 코드는 통보를 보낸다. 열려 진 파일을 호출하는것과 마찬가지로 창문이 실지 열렸는가를 확인하여야 한다. SimpleWindow클라스는 창문의 상태를 되돌리는 GetStatus()성원함수를 가지고 있다. GetStatus()의 원형은 다음과 같다.

```
WindowStatus GetStatus() const  ;
```

여기서 WindowStatus형은 다음과 같다.

```
enum WindowStatus{ Windowclosed, windowopen, WindowFailure};
```

따라서 창문을 열기 위한 보다 안전한 방법은 다음과 같다.

```
HelloWindow.open();
assert(HelloWindow.Getstatus()= =WindowOpen);
```

창문열기가 실패하면 assert마크로는 오유통보를 내보낸다. 통보를 표시하기 위하여 2개의 Simplewindow공개함수를 리용한다. GetCenter()는 창문의 중심자리표를 되돌린다. 두번째 RenderText는 창문의 지정된 위치에 문자렬을 현시한다. 다음의 코드는 창문의 중심에 통보를 내보내게 한다.

```
Position Center = HelloWindow.GetCenter();
Position UpperLeft = Center+Position(-1.0, -1.0);
Position LowerRight = Center+Position(2.0, 1.0);
HelloWindow.RenderText(UpperLeft, LowerRight,
  "Hello EzWindows", White);
```

HelloWindow의 중심자리표가 얻어 지면 본문을 현시하기 위한 4각형의 값이 계산된다. 이 4각형은 오른쪽아래구석과 왼쪽웃구석자리표로 규정할수 있다. 4각형의 중심이 본문의 중심으로 되기때문에 본문은 여기에 현시된다. RenderText()용4각형의 크기는 본문이 4각형의 변두리를 벗어 나지 않도록 계산된다(그림 10-3을 보시오). 결국 한변이 1cm인 통을 리용한다.

RenderText의 세번째 인수는 현시되는 문자렬이며 마지막인수는 본문의 배경색이다. 기정적으로 SimpleWindow창문의 배경색이 흰색이므로 본문은 흰색배경우에 표시된다. EzWindows가 지원하는 색을 다음과 같이 렬거형으로 정의할수 있다.

```
enum color{ black, white, red, green, blue, yellow, cyan, magenta };
```

프로그람의 마지막부분에서는 조작체계가 EzWindows를 통하여 완료통보를 보낼 때 프로그람이 받는 사용자끝통보를 처리한다. 이런 통보는 조종창이 닫길 때 발생한다.

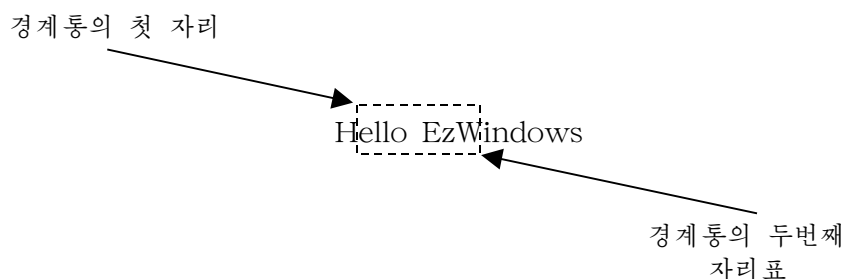



그림 10-3. RenderText()가 리용하는 본문4각형

EzWindows가 완료통보를 받으면 ApiEnd()함수를 호출하여 사용자응용프로그람에 통보를 보낸다. Hello프로그람에서 ApiEnd()는 창문을 닫는 한가지 일만 수행해야 한다. 사용자가 ApiEnd()함수를 작성하지 않는다면 아무 동작도 하지 않는 기정 ApiEnd()함수를 EzWindows서고에서 적재한다. 그림 10-4는 응용프로그람이 실행될 때 창조된 창문을 보여 주며 목록 10-3은 프로그람실행코드를 보여 준다.

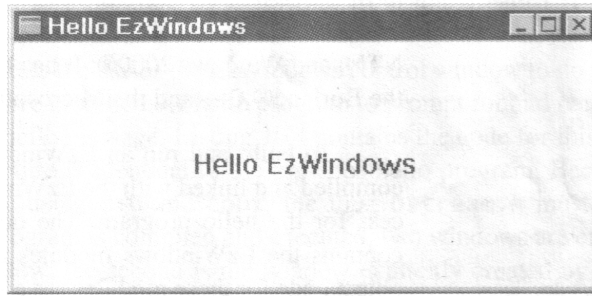



그림 10-4. Hello프로그람이 창조한 창문

SimpleWindow는 마우스와 시간계수기로부터의 사건처리와 튀여나오기통보표시와 같은 추가적인 기능도 가지고 있다. 이러한 문제를 취급하기전에 EzWindows에 포함된 다른 기본클라스 Bitmap를 다시 본다.

10.2.4 EzWindows API기구

이 장에서 론의한 EzWindows API코드와 그에 대한 실례는 책에 부속된 CD-ROM에 제공되여 있다. 프로그람은 World Wide Web주소

http://www.cs.virginia.edu/c++programdesign

에서도 참조할수 있다. 부록 5에 EzWindows API의 개요를 주었다. EzWindows는 Window98, WindowNT, Window2000이 적재된 PC상에서 리용할수 있도록 설계된다.

목록 10-3. Hello.cpp에서 EzWindows Hello의 실현부

```
//Hello EzWindows프로그람
#include "ezwin.h"
#include  <assert.h>
//10*4cm 창문창조
SimpleWindow HelloWindow("Hello EzWindows", 10.0, 4.0, Position(5.0, 6.0);
//ApiMain(): 창문을 창조하고 본문을 현시
int ApiMain(){
    HelloWindow. Open();
    assert(HelloWindow.GetStatus() = = WindowOpen);
    //창문의 중심값얻기
    Position Center = HelloWindow.GetCenter();
    // 본문을 4각형으로 창조
    Position UpperLeft = Center + Position(-1.0, -1.0);
    Position LowerRight = Center + Position(1.0, 1.0);
    //본문현시
    HelloWindow.RenderText(UpperLeft, LowerRight,
     "Hello EzWindows", White);
    return 0;
}
```

```
// ApiEnd(): 창문을 닫는다.
int ApiEnd() {
    HelloWindow.Close();
    return 0;
}
```

Borland C++와 Microsoft Visual C++콤파일러의 최신판을 리용하여 오유수정을 할수 있다.

EzWindows프로그람을 작성하고 실행시키자면 EzWindows코드와 련결되여야 한다. 그림 10-5는 Hello프로그람의 처리단계를 보여 준다. EzWindow API라고 이름을 단 점선4각형에는 프로그람과 련결될 EzWindows모듈들이 있다. 이 모듈에 대한 목적코드는 ezwin.lib서고파일에 정의되였다. C++련결프로그람은 서고로부터 프로그람건설에 필요한 모듈만을 끌어 당긴다. 실행성파일을 만드는 콤파일과정은 다음과 같은 단계를 거친다. 응용프로그람모듈점선통은 프로그람작성자가 작성한 모듈이다.

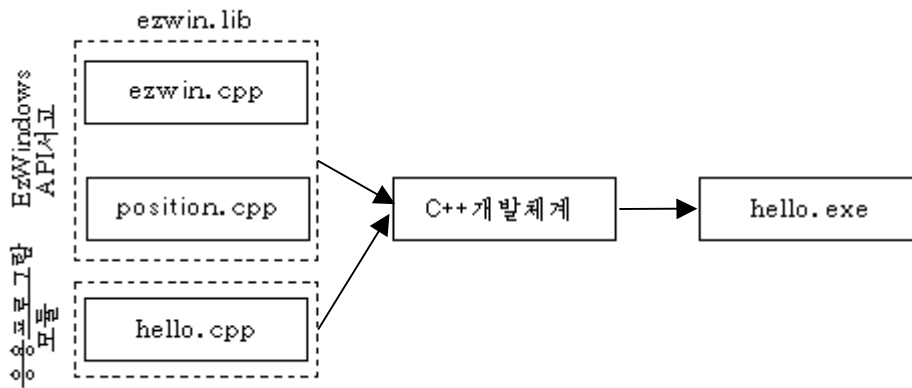



그림 10-5. EzWindows응용프로그람의 작성

대부분의 쏘프트웨어대상과제들이 한개이상의 모듈로 구성되므로 C++프로그람작성환경에는 실행될수 있도록 번역하여야 할 파일들을 규정하는 능력이 있다. 일부 콤파일러들은 대상과제파일(project file)을 리용하며 다른것들은 제작파일(Makefile)을 리용한다. 두 방법에서 다 프로그람작성자는 번역되여야 할 파일, 리용할 서고, 그리고 실행파일의 위치를 규정한다. 부록 6에 각이한 콤파일러들과 가동환경들에서 대상과제파일과 제작파일을 어떻게 설치하는가를 서술하였다.

일단 대상과제파일이 설정되면 해당단추를 누르기하여 실행할수 있는 프로그람을 만들수 있다. 실지 이런 환경에서는 마지막으로 건설된 때부터 파일이 수정되였는지 감시하고 있다. 실행성파일이 만들어진 마지막단계에서는 변경도 할수 있다. 더 큰 프로그람대상과제를 개발하는데서 설정가능한 재콤파일은 많은 시간을 절약할수 있다. 이 장에서 큰 프로그람대상과제는 선택번역을 진행하여 시간을 줄일수 있다. 응용프로그람은 구성하는 프로그람모듈을 보여 주는 그림 10-5에 있는것처럼 도표형식으로 설명하기도 한다.

도형사용자대면부를 리용하는 프로그람에서의 오유수정은 오유정보표시가 쉽지 않으므로 어렵다. EzWindows API는 cout흐름에 삽입된 본문을 현시하는 창문을 항상 열어 놓음으로써 이러한 문제를 해결하였다. 이 창문은 cin흐름에서 추출한 문자를 프로그람에 통보한다. 이 창문을 조종창문(control window)이라고 한다.

EzWindows조종창문의 리용을 보기 위하여 hello프로그람을 hello통보가 나타날 위치를 입력하도록

수정하였다. 목록 10-4에 이 프로그람의 토막을 주었다. 그것의 모듈구조는 앞에서 취급한 Hello프로그람과 차이난다. 프로그람이 cin과 cout를 리용하기때문에 파일 iostream.h가 포함되여야 한 다. 이 프로그람을 번역하여 실행하면 2개의 창문이 창조된다. 하나는 본문을 위한것이고 다른 창문은 프로그람에 의해 창조된 창문이다. 그림 10-6에 그 창문들을 주었다.

목록 10-4. EzWindows프로그람에서 입출력흐름을 례증하는 프로그람

```
//API프로그람의 기능을 설명하는 프로그람
#include <iostream>
#include <string>
#include <assert.h>
#include "ezwin.h"
using namespace std;
//크기가 10*4인 창문을 창조
Simplewindow HelloWindow("Hello EzWindows", 10.0, 4.0,
  Position(5.0, 6.0));
//APIMain() EzWindows의 cin의 리용방법소개
int ApiMain(){
    HelloWindow.Open();
    Assert(HelloWindow.GetStatus() = = WindowOpen);
    cout << "Enter the location in the window¥n"
      << "to write the text (e.g., 4 6): " ;
    int XCoordinate;
    int  YCoordinate;
    cin >> Xcoordinate >> Ycoordinate;
    Position Location(Xcoordinate, Ycoordinate);
      Position UpperLeft = Location + Position (-1.0, -1.0);
    Position LowerRight = Location + Position(1.0, 1.0);
    //본문을 현시
    HelloWindow.RenderText(UpperLeft, LowerRight,
      "Hello EzWindows", White);
    cout << "Text was  rendered at " << Xcoordinate << ","
      << Ycoordinate << end1;
    return 0;
}
// ApiEnd(); 창문을 닫는다.
int ApiEnd() {
    HelloWindow.close();
    return 0;
}
```

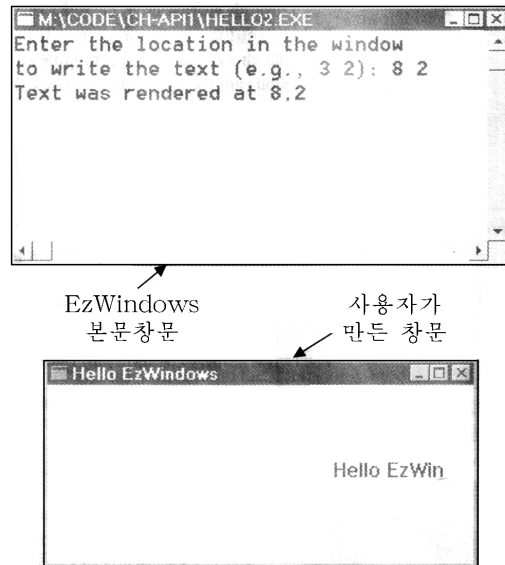



그림 10-6. 목록 10-4를 리용하여 창조한 창문

그림에서 보는것처럼 cout흐름에 삽입된 본문은 본문창에 현시된다. 또한 창문이 능동상태일 때 건반으로 입력한 문자는 cin흐름에 놓이게 된다.

10.3 BitMap클라스

창문체계들은 대체로 도형화상을 현시하는 특성을 가지고 있다. 도형을 보관하는 방법에는 여러가지가 있다. 제일 많이 쓰이는 형식이 비트매프형식(bmp)이다.실례로 많은 색칠하기프로그람(painting program)들은 비트매프파일형식으로 그림을 보관한다. 많은 체계들에서 비트매프파일에 확장자 bmp를 붙여 준다.

비트매프파일을 현시하는 EzWindows의 기본기능을 보여주기 위하여 이 책을 쓴 필자들의 사진을 적재하고 현시하는 프로그람을 작성한다. 이 프로그람에 필요한 모듈은 그림 10-7에 보여 준다.

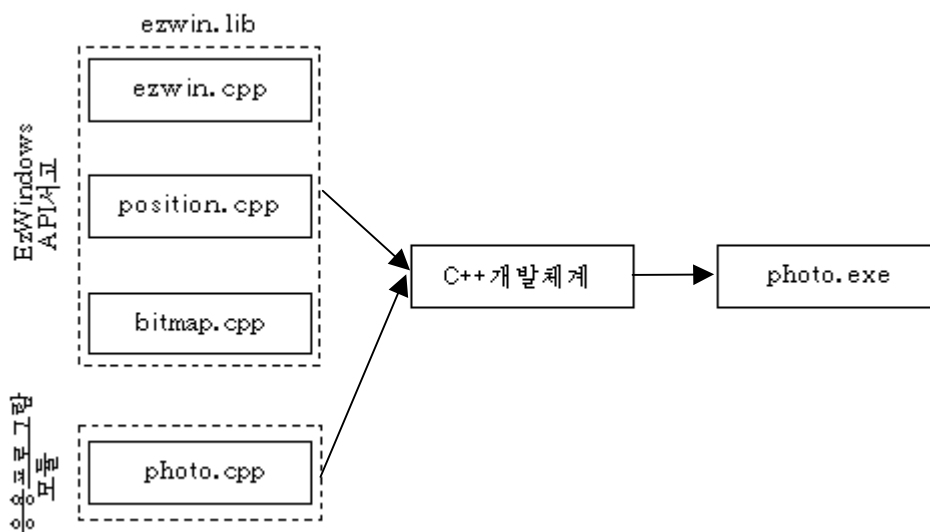



그림 10-7. 사진응용프로그람의 모듈구조

비트매프를 현시하자면 ezwin.cpp, position.cpp, bitmap.cpp모듈이 필요하다. 이 모듈들에는

BitMap클라스의 구체례가 들어 있다. 작성하는 프로그람은 Photo.cpp의 파일에 포함된다. 그림 10-8은 프로그람에 의해 창조되는데 현시된 창문을 보여 준다.

목록 10-3에서는 Hello프로그람에서와 같이 창문을 구체례화할 필요가 있다. Photowindow라는 Simplewindow객체를 창조한다. 이 객체의 전역정의는 다음과 같다.

```
SimpleWindow Photowindow( "The Authors", 10.0, 7.0,
  Position(5.0, 3.0));
```

모든 작업은 ApiMain()에서 진행된다. PhotoWindow가 열려 지며 창문의 중심위치가 얻어 진다. 그다음 PhotoBmp라는 Bitmap객체를 구체례화한다. 정의는 다음과 같다.

```
Bitmap Photo ( PhotoWindow);
```

보는바와 같이 Bitmap구축자는 하나의 묶음인수를 요구한다. 이 인수는 Bitmap가 접속되거나 련결을 가지게 되는 SimpleWindow이다. Bitmap구축자의 선언은 다음과 같다.

```
BitMap ( SimpleWindow &DisplayWindow);
```

이렇게 BitMap객체는 SimpleWindow객체를 참조하여 구축된다.

그림 10-8. 사진프로그람이 창조한 창문

다음단계는 BitMap객체에 화상을 적재하는것이다. 이를 위한 코드를 아래에 보여 준다.

```
Photo.load("Photo.bmp");
Assert(Photo.GetStatus() == BitMapOkay);
```

이 코드는 BitMap객체에 디스크의 사진자료를 적재한다. BitMap성원함수 GetStatus()는 앞에서 취급한 SimpleWindow의 GetStatus와 류사하며 창문의 상대위치를 되돌린다.

Getstatus()의 선언은 다음과 같다.

```
BitMapStatus GetStatus() const ;
```

BitMapStatus의 정의는 다음과 같다.

```
enum BitMapStatus{NoBitMap, BitMapOkay , Nowindow};
```

NoBitMap는 파일이 존재하는가, 안하는가를 지정하며 NoWindow는 BitMap객체와 창문이 련결되

지 않았는가를 가리킨다. BitMap가 구축될 때 BitMap를 포함하는 SimpleWindow객체가 닫긴다면 이러한 상태가 나타나게 된다. 이러한 경우에는 화상이 적재될수 없다. BitMapOkay는 화상이 성과적으로 적재되였는가를 나타낸다.

마지막단계는 PhotoWindow에서 화상을 현시하기 위한 위치를 계산하는것이다. 지금까지 리용한 도형객체와는 달리 BitMap객체는 화상의 왼쪽웃구석의 위치에 놓이게 된다. 다음의 코드는 PhotoBitMap의 적당한 위치를 계산하여 그것이 창문중심에 놓이도록 현시하는 프로그람이다.

```
Position PhotoPosition = WindowCenter +
  Position(-.5, * Photo.GetWidth(),
  -.5 * Photo.GetHeight());
Photo.SetPosition(PhotoPosition);
Photo.Draw();
```

이 코드는 화상의 너비와 높이를 얻기 위해 Getwidth와 GetHeight공개성원함수를 리용한다. 목록 10-5는 Photo.cpp모듈의 코드를 보여 준다.

목록 10-5. BitMap의 적재, 현시

```
//창문의 중심에 저자들의 BitMap사진을 현시한다.
#include "bitmap.h"
#include <assert.h>
//사진을 현시하기 위한 창문을 연다.
Simplewindow PhotoWindow("The Authors", 10.0, 7.0,
  Position(5.0, 3.0));
//BitMap을 현시한다.
int ApiMain() {
    PhotoWindow.Open();
    Assert(PhotoWindow.GetStatus() = = WindowOpen);
    Position windowCenter = PhotoWindow.GetCenter();
    //BitMap을 창조한다.
    BitMap Photo(PhotoWindow);
    //화상을 적재한다.
    Photo.Load("photo.bmp");
    Assert(Photo.GetStatus() = = BitMapOkay);
    //창문의 중심에 대한 자리표 PhotoPosition을 계산한다.
    Position PhotoPosition = WindowCenter +
    Position(-.5 * Photo.GetWidth(),
    -.5 * Photo.GetHeight());
    Photo.SetPosition(PhotoPosition);
    Photo.Draw();
    return 0;
```

```
}
//ApiEnd()창문을 닫는다.
int ApiEnd(){
    PhotoWindow.Close();
    return 0;
}
```

10.4 마우스사건

마우스를 리용하여 콤퓨터를 더 쉽게 조작할수 있다. 마우스는 사용자가 건반으로 지령을 입력하지 않고 프로그람과 직접 호상작용할수 있게 한다. EzWindows는 마우스를 리용하는 간단한 기능도 제공한다. 그 기본원리는 응용프로그람이 EzWindows SimpleWindow에서 마우스누르기사건이 발생할 때 어느 함수를 호출해야 하는가를 EzWindows에 알려 주는것이다. 앞에서 언급한것처럼 이러한 절차를 역호출등록이라고 한다. 마우스사건을 위한 성원함수 SimpleWindow의 선언은 다음과 같다.

```
void SetMouseClickCallback(MouseCallback f);
```

MouseCallback는 typedef형이다.

```
typedef int(*MouseCallback)(const Position &);
```

이 선언은 응용프로그람이 마우스누르기사건에 대하여 역호출(callback)을 등록할 때 const Position을 인수로 받아 들이고 int를 되돌리는 함수의 이름을 주어야 한다는것을 지정하고 있다. 인수의 값은 마우스단추가 눌리웠을 때 마우스지시자의 위치이다. 마우스사건이 어떻게 처리되는가를 설명하기 위해 매 창문에 서로 다른 화상을 현시하는 2개의 창문을 열도록 하는 응용프로그람을 설계하자. 마우스가 창문에 놓여 있고 누르기가 될 때 BitMap는 그 위치에서 창문에 다시 표시된다. 그림 10-9는 프로그람이 실행될 때 어떤 사건이 일어 나는가를 보여 준다.

2개의 전역 SimpleWindow객체를 정의한다. EzWindows로부터 응용프로그람의 통보를 접수할 때 그것들을 해제하고 구체례화하는것이 필요하지 않으므로 2개의 BitMap에 대한 전역정의를 한다. 이 정의는 다음과 같다.

```
SimpleWindow W1("Window One", 15.0, 9.0,
  Position(1.0, 1.0);
SimpleWindow W2("Window Two", 15.0, 9.0,
  Position(8.0, 12.0));
//매 창문에 대한 2개의 BitMap을 정의
BitMap W1Bmp(W1);
BitMap W2Bmp(W2);
```

함수 ApiMain()은 2개의 창문을 열어 BitMap를 적재하고 창문의 기정위치에 BitMap를 표시한다. 우의 코드는 이전에 서술한 프로그람과 비슷하다(목록 10-5에 상세하게 설명). ApiMain()의 마지막동작은 마우스사건의 특성을 등록하는것이다. 매 창문에 대한 등록기능이 필요하다. 이 기능을 수행하기 위한 코드는 다음과 같다.

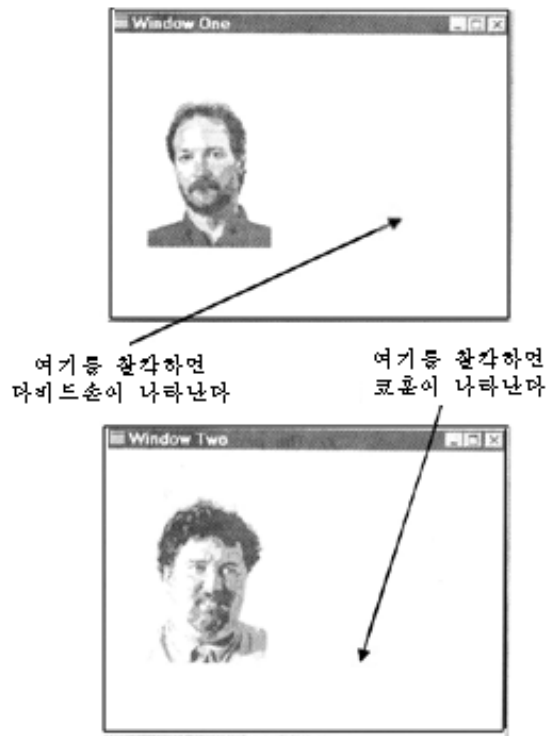



그림 10-9. 마우스사건프로그람에 의해 창조된 창문

```
W1.SetMouseClickCallback(W1MouseClick);
W2.SetMouseClickCallback(W2MouseClick);
```

이 코드는 마우스사건이 접수될 때 함수 W1MouseClick를 호출하여 W1에게 알려 준다. 이 코드는 W2에 류사한 통보를 보낸다. W1MouseClick의 정의는 다음과 같다.

```
int W1MouseClick(const Position &p){
    //BitMap을 지운다.
    W1Bmp.Erase();
    //새 위치를 설정하고 BitMap을 현시한다.
    W1Bmp.SetPosition(p);
    W2Bmp.Draw();
    return 1;
}
```

W1의 마우스사건처리기는 련관된 비트매프를 지우고 EzWindows로부터 입력한 값으로 화상의 위치를 재설정한다. 그리고 새 위치에 비트매프(bitmap)를 그린다. 이 코드는 mevent.cpp모듈에 들어 있다. 그림 10-10에 모듈구조를 보여 주었다.

이 모듈을 콤파일하여 련결하면 mevent.exe 파일이 만들어 진다. W2MouseClick에 대한 코드도 이와 류사한데 목록 10-6에 보여 주었다.

목록 10-6. 마우스사건에 응답하는것을 보여 주는 프로그람

```
//마우스의 리용을 보여 주는 실례
//마우스가 눌리워 진 곳에서 BitMap를 현시하는 프로그람
//2개의 창문을 정의
#include "bitmap.h"
#include <assert.h>
//2개의 서로 다른 창문정의
SimpleWindow W1("Window One", 15.0, 9.0,
  Position(1.0, 1.0));
SimpleWindow W2("Window Two", 15.0, 9.0,
  Position(8.0, 12.0));
//매 창문에 대한 2개의 BitMap정의
BitMap W1Bmp(W1);
BitMap W2Bmp(W2);
//W1mouseClick(): 창문1의 역호출
int W1MouseClick(const Position &p){
    //BitMap지우기
    W1Bmp.Erase();
    //새 위치를 설정하고 BitMap 현시
    W1Bmp.SetPosition(p);
    W1Bmp.Draw();
    return 1;
}
//W2MouseClick(): 창문2의 역호출
int W2MouseClick( const Position &p){
    //BitMap지우기
    W2Bmp.Erase();
    //새 위치를 설정하고 BitMap 현시
    W2Bmp.SetPosition(p);
    W2Bmp.Draw();
    return 1;
}
int ApiMain(){
    // 창문들을 연다.
    W1.Open(0;
    Assert(W1.GetStatus() = = WindowOpen);
    W2.Open();
    Assert(W2.GetStatus() == WindowOpen);
    //화상을 적재한다.
```

```
    W1Bmp.Load("c1.bmp");
    Assert(W1Bmp.GetStatus() == BitMapOkay);
    W2Bmp.Load("c2.bmp");
    Assert(W2Bmp.GetStatus() = = BitMapOkay);
    //시작위치에 BitMap 현시
    W1Bmp.SetPosition(Position(1.0, 1.0));
    W2Bmp.SetPosition(Position(1.0, 1.0));
    W1Bmp.Draw();
    W2Bmp.Draw();
    //매 창문에 대한 등록기역호출
    W1.SetMouseClickCallback(W1MouseClick);
    W2.SetMouseClickCallback(W2MouseClick);
    return 0;
}
int ApiEnd(){
    //창문을 닫는다.
    W1.Close();
    W2.Close();
    return 0;
}
```

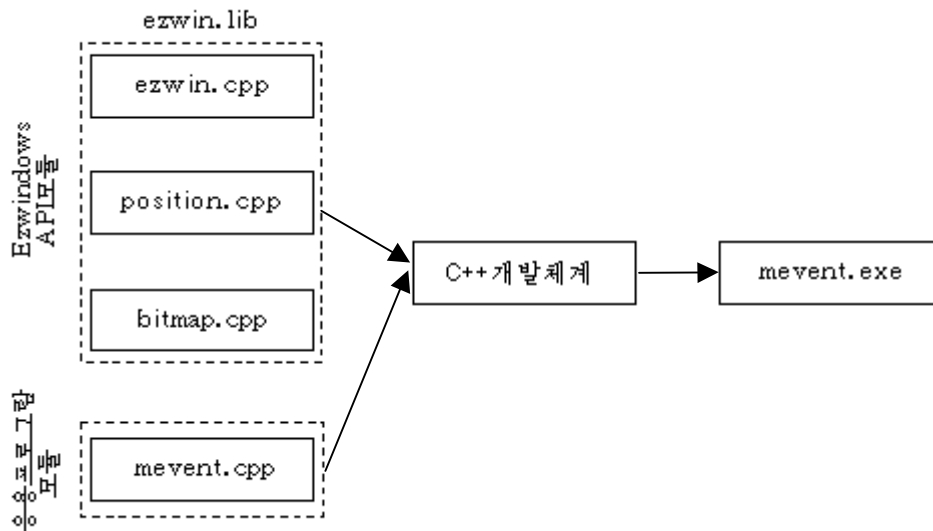



그림 10-10. 마우스사건응용프로그람의 모듈구조

10.5 비트매프와 마우스사건

BitMap의 쓸모 있는 특성은 마우스위치가 비트매프내에 있는가를 결정하는 능력이다. EzWindows BitMap는 이러한 능력을 제공해 준다. 이 성원함수의 선언은 다음과 같다.

```
bool Bitmap::IsInside(const Position &AtPosn);
```

이 선언은 AtPosn이 비트매프를 포함하면 True를 되돌리고 그렇지 않으면 False를 되돌린다. 이 능력은 마우스와 접속하여 간단한 조종을 설계하기 위한 특성을 제공한다.

실례로 취해야 할 어떤 동작을 표현하는 Bitmap를 현시할수 있다. 그 비트매프내에 마우스유표를 놓고 단추를 누름으로써 이 동작이 수행되게 할수 있다. 이 능력을 보여 주기 위하여 카드의 비트매프화상을 현시하는 프로그람을 작성하자. 마우스가 그 카드내에서 누르기되면 카드는 절환된다. 이 응용프로그람코드는 flip.cpp파일에 포함되여 있으며 그림 10-11에 이 프로그람에 필요한 모듈을 보여 주었다.

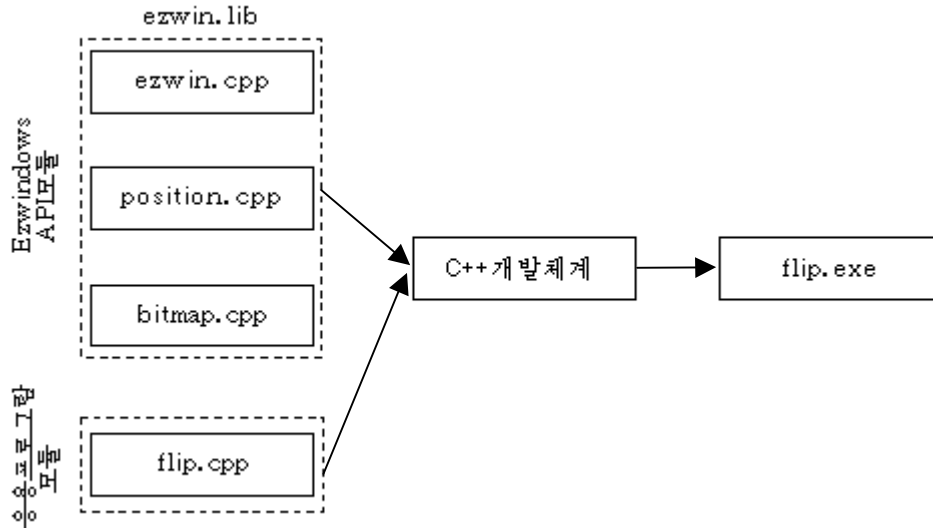



그림 10-11. flip.cpp응용프로그람의 모듈구조

앞에서 취급한 EzWindows의 표본프로그람과 같이 창문을 정의한다. 이때 두개의 비트매프자료가 필요하다. 하나는 카드의 앞면이고 다른 하나는 뒤면이다. 이런 전역객체들외에 카드의 어느 면이 보여지는가를 기억하는 객체가 필요하다. 이 객체를 상태변수(state variable)라고 한다는데 대해서 상기하자. 이에 대한 정의를 아래에 주었다.

```
//창문을 정의
SimpleWindow FlipWindow("FlipCard", 15.0, 9.0,
  Position(1.0, 1.0));
//카드의 정면과 뒤면에 대한 BitMap를 정의
BitMap CardFront(FlipWindow);
BitMap CardBack(FlipWindow);
//보여 주는 카드의 측면을 기억하는 객체와 형을 요구
enum Side { Front, Back };
Side SideShowing;
```

ApiMain()함수는 이전에 서술한 코드들과 류사하다. 여기서 새로운것은 마우스역호출함수(mouse callback function)안에서 무엇이 일어 나는가 하는것이다. 프로그람이 통보를 접수하면 프로그람은 마우스가 카드내부를 지적하고 있는가를 검사한다. 이 함수의 구체례는 다음과 같다.

```
int MouseClickEvent( const Position &MousePosition){
    if (CardFront.IsInside(MousePosition)) {
        //카드가 설정된다.
```

```
            if (SideShowing == Back) {
                SideShowing = Front;
                CardFront.Draw();
            }
            else if (SideShowint = = Front) {
                SideShowing = Back;
                CardBack.Draw()
            }
        }
        return 1;
    }
```

마우스가 카드안에서 눌리워 졌다면 프로그람은 상태변수를 검사한다. 현재 보이는것이 뒤면이라면 프로그람은 카드앞면을 그리며 SideShowing을 Front로 바꾼다. 만일 현재 보이는것이 앞면이라면 카드의 뒤면을 그리고 SideShowing을 Back로 바꾼다. 화면에서 마우스가 카드내부를 누르기할 때마다 카드가 뒤집어 진다. 목록 10-7에 이 기능을 수행하는 코드를 주었다.

목록 10-7. 마우스가 객체를 지적하고 있는가를 검사하는 프로그람

```
//BitMap를 누르고 설정하는 동작을 보여 준다.
#include <assert.h>
#include "bitmap.h"
//창문을 정의
Simplewindow Flipwindow("FlipCard", 15.0, 9.0,
  Position(1.0, 1.0));
//카드의 앞면과 뒤면에 대한 BitMap를 정의
BitMap CardFront(FlipWindow);
BitMap CardBack(FlipWindow);
//보여 지는 카드의 면을 기억하는 객체와 형을 요구
enum Side { Front, Back };
Side SideShowing;
//MouseClickEvent(): 사용자가 마우스를 누를 때 이 함수가 호출된다.
int MouseClickEvent(CardFront.IsInside(MousePosition)) {
    if (CardFront.IsInside(MousePosition)) {
        //카드가 설정된다.
        if (SideShowing  == Back) {
            SideShowing = Front;
            CardFront.Draw();
        }
        else if (SideShowing = = Front) {
```

```
                SideShowing = Back;
                CardBack.Draw();
            }
        }
        return 1;
    }
    int ApiMain() {
        //창문을 연다.
        Flipwindow.Open();
        Assert(FlipWindow.GetStatus() = = WindowOpen);
        //화상을 적재한다.
        CardFront.Load("c1.bmp");
        Assert(CardBack.GetStatus() = = BitMapOkay);
        //카드를 현시하기 위한 위치계산
        Position CardPosition = FlipWindow.GetCenter() +
          Position(1.5 * CardFront.GetWidth(),
          -.5 * CardFront.GetHeight());
        CardFront.SetPosition(CardPosition);
        CardBack.SetPosition(CardPosition);
        SideShowing = Front;
        CardFront.Draw();
        //마우스역호출을 설정
        Flipwindow.SetMouseClickCallback(MouseClickEvent);
        return 0;
}
```

10.6 시간계수기사건

EzWindows의 다른 특징은 시간계수기설정능력이다. 시간계수기(timer)는 미리 결정된 시간이나 간격으로 어떤 동작을 수행하는 프로그람을 작성할 때 아주 유용하다. 시간계수기를 EzWindows에 의해 관리되는 자명종시계로 상상할수 있다. 자명종시계가 꺼지게 되면 EzWindows는 역호출함수를 통해 통보를 응용프로그람에 보낸다. 시간을 설정하고 관리하기 위한 Simplewindow성원함수는 SetTimerCallBack(), StartTimer()와 Stoptimer()이다. SetCallBack()는 SetMouseClickCallBack()와 비슷하다. 이 함수는 사용자의 역호출함수를 등록한다. 역호출함수는 시간계수기사건이 일어 날 때 호출된다.

성원함수 StartTimer()는 시간계수를 시작하게 한다. 이 함수는 하나의 인수를 가지며 시간계수기사건이 어느 정도 자주 발생하는가를 규정한다. 그 인수는 미리초(ms)로 된다. 실례로 명령

```
Swin.SettimerCallback(TimerHandler);
Swin.StartTimer(1000);
```

은 SimpleWindowSwin에 대해 시간계수기를 설정한다. 시간계수기는 1000ms에 한번씩 동작한다. 그러면 사용자의 TimerHandler()보조프로그람이 호출된다. 시간계수기사건이 더이상 요구되지 않을 때 시간계수기 StopTimer()를 호출하여 시간계수기를 중지시킨다. 그 명령문은 다음과 같다.

```
Swin.StopTimer();
```

이 명령은 시작된 시간계수기를 멈추게 한다. 시간계수기사건의 리용을 설명하기 위하여 10.4에서 설명한 Mouse.cpp를 변경하여 1/2초에 한번씩 창문에 현시할 화상의 위치를 란수적으로 발생시켜 보자. 그림 10-12는 이 실례프로그람의 모듈구조를 보여 준다.

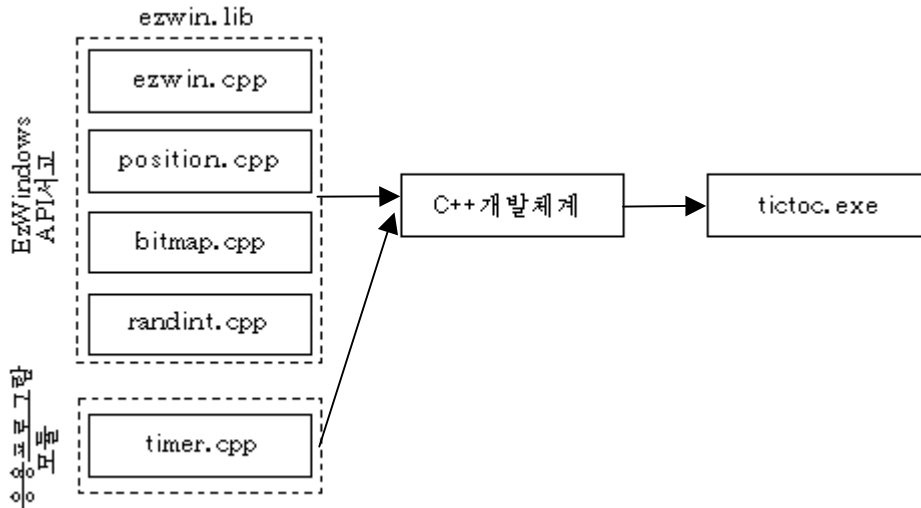



그림 10-12. 시간계수기응용프로그람의 모듈구조

새로운 모듈은 Timer.cpp라고 부르지만 대부분의 코드들은 Mouse.cpp와 같다. 2개의 마우스역호출함수를 설정하는것대신 사건역호출함수를 2개 설정하였다. 2개의 창문에 대한 역호출을 설정하는 명령문은 아래와 같다.

```
W1.SetTimerCallback(W1TimerEvent);
W2.SetTimerCallback(W2TimerEvent);
Bool TimerStatus1 = W1.StartTimer(500);
Bool TimerStatus2 = W2.StartTimer(500);
Assert(TimerStatus1 && Timerstatus2);
```

시간계수기사건역호출함수는 다음과 같다.

```
int W1TimerEvent() {
    Redisplay(W1, W1Bmp);
    return 1;
}
int W2TimerEvent() {
    Redisplay(W2, W2Bmp);
    return 1;
}
```

함수 ReDisplay()는 비트매프의 표시위치를 우연적으로 처리한다. 목록 10-8은 이 시간계수기사건에 대한 프로그람을 보여 준다.

목록 10-8. 시간계수기사건을 례증하는 프로그람

```
//시간계수기사건의 리용을 보여 주는 실례
//프로그람은 시간계수기를 동작시켜 우연위치에 비트매프를 현시한다.
#include <assert.h>
#include "bitmap.h"
#include "randint.h"
//2개의 창문을 정의한다.
Simplewindow W1("Window One", 15.0, 9.0,
  Position(1.0, 1.0));
SimpleWindow W2("Window Two", 15.0, 9.0,
  Position(8.0, 12.0));
//매 창문에 대한 2개의 비트매프를 정의한다.
BitMap W1Bmp(W1);
BitMap W2Bmp(W2);
//ReDisplay(): 새 위치에 비트매프를 옮긴다.
void Redisplay(SimpleWindow &W, BitMap &B){
    //BitMap 지운다.
    B.Erase();
    //새 위치를 계산하고 비트매프를 현시한다.
    //창문에 비트매프가 현시되였는가를 확인한다.
    //란수발생을 초기화한다.
    //그다음 X자리표에 대한 란수를 발생시킨다.
    EzRandomize();
    Randomint X(1, (int) w.GetWidth());
    int XCoord = X.Draw();
    if (XCoord + B.GetWidth() > W. GetWidth())
        XCoord = XCoord - B. GetWidth();
//Randomint객체를 Y위치에 창조
    Randomint Y(1. (int) w.GetHeight());
    int YCoord = Y.Draw();
    if (YCoord + B.GetHeight() > W. GetHeight())
        YCoord = YCoord - B. GetHeight();
    B.SetPosition(Position(Xcoord, Ycoord));
    B.Draw();
}
//W1TimerEvent(): 창문1에 대한 역호출함수
```

```
int W1timerEvent(){
    Redisplay(W1, W1Bmp);
    return 1;
}
//W2TimerEvent(): 창문2에 대한 역호출함수
int W2Timerevent() {
    Redisplay(W2, W2Bmp);
    return 1;
}
//ApiMain(): 창문을 열고 시간계수기를 기동
int ApiMain() {
    //창문을 연다.
    W1.Open();
    Assert(W1.GetStatus() = = WindowOpen);
    W2.Open();
    Assert(W2.GetStatus() = = windowOpen);
    //화상을 적재한다.
    W1Bmp.Load("c1.bmp");
    Assert(W1Bmp.GetStatus()  = = BitMapOkay);
    W2Bmp.Load("c2.bmp");
    Assert(W2Bmp.GetStatus() = = BitMapOkay);
    //시작위치에 비트매프를 현시한다.
    W1Bmp.SetPosition(Position(1.0, 1.0));
    W2Bmp.SetPosition(Position(1.0, 1.0));
    W1Bmp.Draw();
    W2Bmp.Draw();
    //매 창문에 대한 역호출을 등록하고 50ms에 한번씩 시간계수기를 기동
    W1.SetTimerCallback(W1TimerEvent);
    W2.SetTimerCallback(W2TimerEvent);
    Bool timerStatus1 = W1.StartTimer(500);
    Bool timerStatus2 = W2.StartTimer(500);
    Assert (TimerStatus1 && TimerStatus2);
    return 0;
}
int ApiEnd() {
    //시간계수기를 정지하고 창문을 닫는다.
    W1.StopTimer();
    W2.StopTimer();
    W1.Close();
```

```
    W2.Close();
    return 0;
}
```

10.7 경보통보문

창문이 출현하여 사용자가 무시할수 없는 통보문을 표시하는 일이 자주 생긴다. 이 통보가 무시되지 않는가를 확인하는 한가지 방법은 응용프로그람이 열기한 창문에서의 작업으로부터 사용자를 분리하는것이다. 이러한 창문을 흔히 경보(alert)창문 혹은 방식형대화칸(modal dialog box)이라고 한다. 클라스 SimpleWindow는 경보창문을 현시하는 기능을 수행한다. 실례로 Jitterbug라는 SimpleWindow가 열리면 명령문

```
Jitterbug.message("Nice swatting!");
```

은 경보창문이 나타나게 한다. 사용자는 대화칸이 사라질 때까지 응용프로그람에 소속된 임의의 창문에서 아무런 작업도 할수 없다(프로그람실행은 대화칸이 사라질 때까지 정지된다). 대화칸의 동작을 설명하기 위하여 Hello프로그람을 변경시켜서 본문이 현시되기전에 대화칸이 나타나도록 하자. 그림 10-13에 이 모듈구조를 보여 준다.

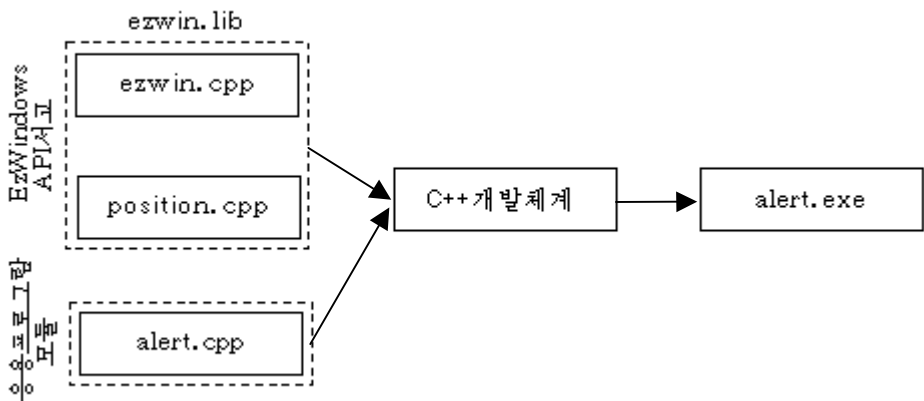



그림 10-13. 경보응용프로그람의 모듈구조

수정된 ApiMain()은 다음과 같다.

```
int ApiMain() {
    HelloWindow.Open();
    Assert(HelloWindow.GetStatus() = = WindowOpen);
    //창문의 중심값을 얻는다.
    Position Center = HelloWindow.GetCenter();
    //본문을 포함하는 4각형을 창조한다.
    Position UpperLeft = Center + Position(-1.0, -1.0);
    Position LowerRight = Center + Position(1.0, 1.0);
    HelloWindow.Message("Click Ok to continue");
    //본문을 현시한다.
```

```
        HelloWindow.RenderText(UpperLeft, LowerRight,
        "Hello EzWindows", White);
        return 0;
    }
```

프로그람은 그림 10-14에서 보여 주는것처럼 창문을 창조한다. 기본창문은 본문을 현시하지 않는다. 마우스가 기본창문안에서 누르기된다면 프로그람이 정지되였다고 사용자에게 경보음을 울린다. 사용자가 OK를 누를 때 경보칸은 사라지며 본문 Hello EzWindows가 창문에 나타난다.

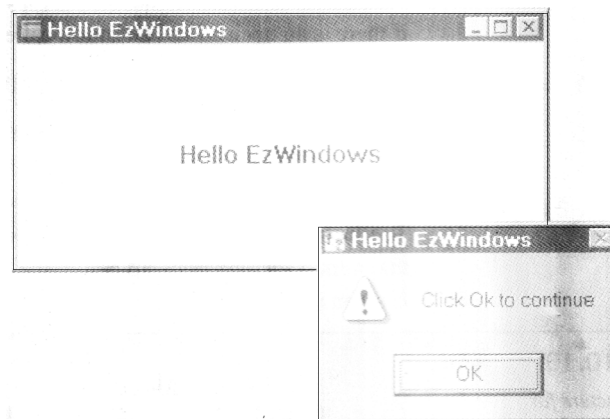
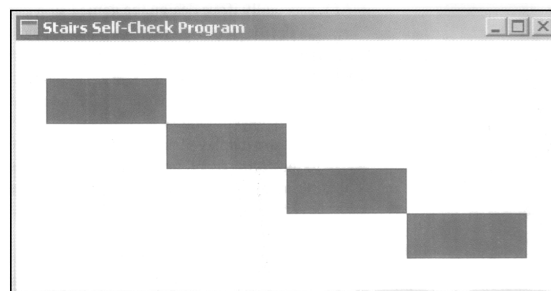



그림 10-14. 창문과 함께 표시된 통보창문

문 제

1. 어떤 사건이 일어 날 때 호출하는 루틴을 설정하는 처리는 무엇인가?
2. 마우스가 SimpleWindow내에서 누르기될 때 유표의 자리표를 현시하는 ReportMousePosition()함수를 작성하시오. 유표의 자리표를 현시하는 함수 ReportMousePosition()을 리용하는 driver프로그람을 작성하여 이 함수를 실행시켜 보시오.
3. BitMap파일의 이름을 접수하는 EzWindows프로그람을 작성하시오. 프로그람은 창문의 중심에 BitMap자료를 현시해야 한다.
4. SimpleWindow에서 비트매프를 현시하는 EzWindows프로그람을 작성하시오. 마우스가 비트매프를 누르면 비트매프가 설정되였다는 경보통보문이 나타난다. 만일 마우스가 창문의 밖에서 누르기되였다면 비트매프가 설정되지 않았다고 경보통보문이 나타난다.
5. 창문에 4각형통을 그리는 EzWindows프로그람을 작성하시오. 프로그람은 다음의 그림과 같은 도형을 현시해야 한다.

6. 학습장종이에 뚫려 진 구멍크기만한 흑색원의 비트매프를 창조하시오. Paint 혹은 PhotoShop와 같

은 프로그람을 리용할수 있다. SimpleWindows의 우연적인 위치에 구멍을 내는 Punch라는 EzWindows프로그람을 작성하시오. 이 구멍뚫기프로그람의 속도는 EzWindows의 시간계수기에 의해 조종된다.

7. SimpleWindow에 마우스로 직선을 그리는 프로그람을 작성하시오. 부록 5에 있는 EzWindow의 클라스인 Raysegment를 리용하시오.
8. 입력재촉상태에서 옹근수 1부터 9사이를 입력하는 프로그람을 작성하시오. 그다음 프로그람은 창문에 《countdown》이라고 현시해야 한다. 시간계수기를 리용하여 매초마다 표시가 달라 지게 한다. 수자 BitMap는 부속 CD-ROM에 있는 EzWindow등록부에서 찾아 리용하시오.

10.8 Simon says유희

Simon says라는 어린이유희를 생각해 보자. 이 유희를 개발하는것으로써 EzWindows API에 대하여 결론 짓게 된다. Simon이라고 부르는 이 유희에서는 카드모임이 창문안에 표시된다. 유희는 콤퓨터가 우연순서로 카드들을 배치하면 시작한다. 유희자는 그것들을 동일한 순서로 배치하여야 한다. 배치하면 다른것이 표시된다. 유희는 유희자가 실패하든지 아니면 성공할 때까지 계속한다.

객체지향원리들을 리용하고 있는 유희를 설계한다면 유희작성은 EzWindow API를 리용할 때 실제적으로 그리 어렵지 않다. 먼저 어느 객체가 필요하며 다른것과 어떻게 통보를 협조하는가를 결정하여야 한다.

Simon유희는 여러 형식의 화상을 표시하는 창문으로 구성된다. 유희의 보다 형식적인 내용은 첫 과제를 방조할것이다. 화상의 한 형태는 번질수 있는 카드이다. 4개의 카드들은 창문을 가로 건너 서로 건설하여 표시된다. 카드들외에 조종화상들이 있다. 한 화상은 재시작단추이다. 마우스가 이 단추로 누르기할 때 유희는 시작하기표식에서 시작한다. 또 다른 화상은 포기단추이다. 그것이 선택될 때 유희는 끝난다.

Simon은 다음과 같이 진행된다. 카드들은 우연적으로 잠시 다쳐 진다. 처음에 세개 카드들이 다쳐 진다. 결과가 보여 진후 유희자는 동일한 순서에서 카드들을 선택해야 한다. 유희자가 새 결과에 성공이면 전것보다 한 카드는 더 길게 된다. 유희자는 6개 카드들이 성공적으로 재호출될 때 유희에서 이긴다.

선행하고 있는 설명으로부터 카드들을 표현하는 객체가 요구되는것은 명백하다. 카드들의 한 형태는 그것들이 다쳐 지는것이 알수 있게 되여야 한다는것이다. 또한 마우스가 카드에서 지적하는지 안하는지를 결정하여야 한다. 우리는 이 클라스 Simon객체를 호출한다. 그것은 EzWindows Bitmap가 Simon객체에 의해 요구한 훨씬 더 많은 기능적인것을 제공한다는것을 명백히 해둔다. 또 하나의 류사한 객체들은 조종객체들이다. 이것들은 또한 EzWindows BitMap객체들에 의해 표현된다.

요구되는 서로 다른 객체는 때때로 유희경기를 조종하는것이다. 그것은 적당한 순서에서 다쳐 지게 될 카드들을 일으키고 결과를 산생시키기 위한 책임일것이며 그때 유희자가 정확한 결과를 재호출하는지 안하는지를 보기위해 검사하고 있을것이다. 그것은 또한 유희를 설정하고 유희가 끝나는것을 결정하기 위한 책임이다. 우리는 이 클라스 SimonGame을 호출한다..

또 하나의 중요한 객체는 우리의 서술유희자안에서 언급되였다. 다행히 유희자를 실행하는것이 필요치 않기때문에 이 함수는 우리들이 가지고 있는 기능만을 제공된다.

유희안에서 객체들은 그림 10-15에서 보여 준것으로서 모듈안으로 프로그람의 자연적인 분구를 제공한다. 모듈 simobj.cpp는 클라스 SimonObject의 실행을 포함하며 모듈 simobj.cpp는 클라스

SimonGame의 실행을 포함한다. 함수 ApiMain()과 마우스 그리고 시간계수기역호출은 모듈 simobj.cpp안에 있다.

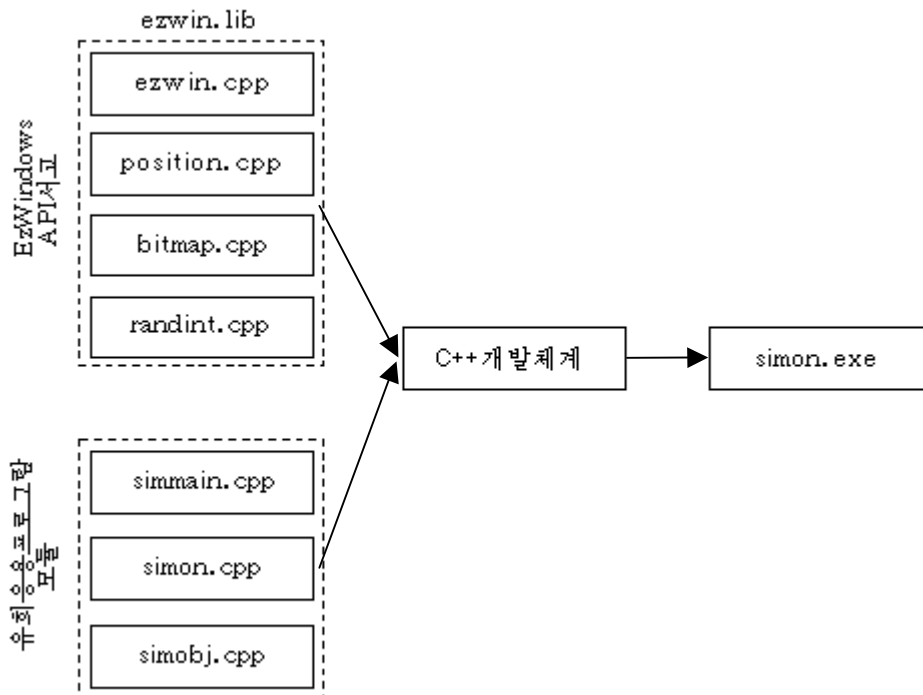



그림 10-15. 유희모듈구조

SimonGame은 유희를 조종하기 위해 포함되는 속성들과 객체들이 수행되는데서 대략적인 지름을 얻는다. SimonGame에 의해 제공한 형태들과 동작들은 창문안에 유희판설치하기, 카드들을 다치기 위해 결과를 산생시키기, 카드다치기, 창문안에서 마우스누르기들을 가지고 관계한다. 마우스누르기는 유희를 재시작 혹은 탈퇴인가를 질문하고 다음에 나타나는 하나로서 선택되고 있는 유희자를 의미하는 카드내부에 있어야 한다. 유희자가 유희에서 실패하든지 아니면 성공할 때까지 유희는 계속된다. 객체지향원리를 리용하여 유희를 설계하면 즉 EzWindows API를 리용할 때 유희작성은 그리 어렵지 않다. 먼저 어느 객체가 필요하며 다른것들과 어떻게 통보를 협조하는가를 결정할 필요가 있다. 첫번째 과제가 처리되면 유희의 형식상들이 마련될것이다.

Simon유희는 여러 형식의 화상들이 표시되는 창문으로 구성된다. 화상의 한 형식은 뒤번져 질수 있는 카드이다. 4개 카드는 창문을 건너 다른것과 이웃하여 표시된다. 카드외에 조종과 관련된 화상들이 있다. 하나는 재시동단추이다. 마우스가 이 단추를 누르면 유희가 초기수준에서 시작한다. 다른것은 완료단추이다. 그것이 선택되면 유희는 완료된다. Simon은 다음과 같이 동작한다. 카드들은 우연적인 순서로 뒤집어 져 있다. 초기에 3개 카드가 뒤집어 져 있다. 렬이 구성되면 유희자는 같은 규칙으로 카드들을 선택해야 한다. 성공하면 새렬이 이전보다 하나 더 많이 구성된다. 6개 카드렬에 대하여 성공하면 유희에서 이긴다.

이상으로부터 카드를 표현하는 객체가 필요하다는것은 명백하다. 카드의 한가지 동작은 번져 질수 있어야 한다는것이다. 이를 위하여 SimonObject라는 클라스를 정의하기도 한다. 명백한것은 EzWindows Bitmap클라스가 SimonObject에 필요한 기능을 거의 다 제공한다는것이다. 한편 근사한 객체는 조종객체들이다. 이것들 역시 EzWindows Bitmap객체에 의해 표현된다.

또 다른 객체는 유희를 조종하는것이다. 이 객체는 렬을 구성하고 적당한 규칙으로 카드들을 번질수

있어야 하며 유희자가 정확한 렬로 재구성하였는가 검사할수 있어야 하는것이다. 또한 오락이 끝났는가를 결정할수도 있어야 한다. 이 클라스를 SimonGame이라고 하기로 한다.

다른 중요한 객체 즉 player가 해설에서 언급된다. 다행히 이 기능을 유희가 제공하므로 player를 실현할 필요는 없다.

유희에 있는 객체들은 또한 그림 10-15와 같이 모듈들에 프로그람의 원래분구를 제공한다. 모듈 Simobj.cpp에는 SimonObject클라스의 구체례가 있고 Simon.cpp에는 SimonGame클라스구체례가 들어있다. ApiMain()함수와 마우스, 시간계수기역호출함수는 Simmain.cpp에 있다.

SimonGame은 유희를 조종하므로 어떤 속성과 객체를 가져야 하며 어떤 일을 하여야 하는가를 대충 보기로 한다. SimonGame이 제공한 기능은 창문에 유희판을 설정하고 카드의 렬을 만들고 카드를 번지고 창문에서 마우스누르기를 취급하는것이다.

마우스조종과 관련한 비트매트프중 어느 하나를 누르기하는데 이에 따라 유희를 재시동하겠는가 완료하겠는가를 문의하게 되며 카드를 누르면 렬에서 다음으로 주목되였던 카드가 선택된다는것을 의미한다. SimpleGame의 동작과 기능은 다음과 같다.

- 유희를 초기화한다.
- 마우스누르기사건을 처리한다.
- 카드렬을 만든다.
- 유희를 관리한다.
- 카드를 번진다.

명백하게 SimonGame에는 유희속성이 모두 포함되여야 한다. 이 속성들외에 SimonGame에는 SimonObject를 표시하기 위한 SampleWindow 그리고 유희를 조종하기 위한 비트매프 등이 있다. 유희를 조종하기 위한 비트매프는 Restart와 Quit로 부르기로 한다. SimonGame의 속성은 다음과 같다.

- 유희를 위한 SimpleWindow 객체
- Simon객체들의 묶음
- 재시작과 탈퇴, 비트매프들
- 카드를 번지는 순서
- 그것이 돌리도록 포함하는 상태특성
- 현재렬의 길이

클라스 SimonGame흐름들의 첫 판본은 다음과 같다.

```
class SimonGame {
    enum Turn { Simon, Player };
    public:
        SimonGame(SimpleWindow &Window);
        void Initialize();
        void Play();
        void MouseClick(const Position &MousePosn);
        int Timer();
        void Pickorder();
    private:
```

```
    SimpleWindow &w;
    Vector<SimonObject> Posn;
    BitMap Restart;
    BitMap Quit;
    Vector<int > Order;
    Turn WhoseTurn;
    int SequenceLength;
};
```

먼저 성원자료들을 보기로 하자. 이전에 리용하던 모든 객체들이 창문을 포함하는것처럼 SimonGame은 유희를 표시하는 창문에 대한 참조를 가진다. 이 객체들은 창문에 표시되며 우선적인 순서로 놓여 있다. Posn의 크기는 SimonGame객체가 구축되면서 유희가 초기화될 때 설정될것이다. Restart와 Quit는 유희를 조종하기 위한 비트매프들이다. 벡토르 Order는 Simon객체를 다치는 결과를 포함한다. 상태객체 WhoseTurn은 콤퓨터인가 유희자인가를 지적한다. 마지막으로 자료성원 Sequencelength는 산생하려는 결과길이를 포함한다. 유희가 진척될수록 이 객체들은 더 커진다.

공개성원함수들가운데는 구축자가 있는데 구축자는 입력으로서 유희를 표시하는 창문을 가진다. Play()는 유희를 초기상태로 하며 그다음 시작시킨다. MouseClick()과 Timer()는 마우스가 눌린 때와 시간계수기사건때 각각 호출된다. 이 함수들은 유희의 많은 문제들을 처리한다. Pickorder()는 카드를 번지는 우연순서를 생성한다. Initialize()는 유희판을 설치한다. 이 함수들은 더 많은 유희론리들을 조종한다. PickOrder()는 카드들을 다치는 우연결과를 산생한다. Simonobject의 선언은 다음과 같다.

```
Enum side { Front ,Back };
class SimonObject {
    public:
        SimonObject();
        void Initialize(SimpleWindow &GameWindow,
        const string &FrontFile, const string
&BackFile,
        const position &Posn);
        void setSide(const side &s);
        void Draw();
        void Flip ();
        bool InInside(const position &MousePosition)
         const;
    private:
        Bitmap FrontSide;
        Bitmap Backside;
        Side SideShowing;
};
```

앞서 본 동작들외에 SimonObject는 성원자료 SideShowing을 위한 검토자와 변이자를 가진다. 중

요한 2개 객체에 대한 사상을 가질수 있도록 프로그람이 어떻게 동작할것인가를 설명한다. 판대기는 SimonObject들 몇개와 조종단추들로 구성된다. 이것들은 EzWindows Bitmap들이다. Bitmap의 IsInside성원함수를 리용하여 SimonObject나 조종단추를 마우스가 지적하고 있는가를 결정할수 있을것이다. 유희의 흐름은 시간계수기사건과 마우스누르기사건에 의하여 조종되는데 이 사건들은 SimonGame에 의하여 처리된다. 그림 10-16은 클라스들과 그것들사이에 보내지는 통보들이다. 도표에서 보는것처럼 Bitmap클라스는 유희용블로크를 건설하는 열쇠이다.

유희는 세 단계를 거친다. 초기화단계에서는 판을 설정하고 유희를 진행할수 있는 조건을 준비한다. 두번째 단계에서는 SimonObject들을 배렬하고 세번째 단계에서 유희자는 SimonObject들을 선택한다. 세번째 단계는 또한 사용자의 선택이 옳은가를 검사한다.

첫 단계는 명백하다. 먼저 그것을 작성하고 요구하는대로 보이는가를 검사한다. 첫번째것은 SimonGame용구축자를 작성하는것이다.

```
SimonGame::SimonGame(SimpleWindow &Window) :
  W(Window) {
    //simonobject의 공간과 다쳐 지는 객체의 순서를 반전
    posn.reserve(MaxPositions);
    order.reserve(MaxSequencelength);
}
```

앞부분의 코드는 성원자료 w를 초기화하고 벡토르안에 필요공간을 예약한다. 모든 실제작업은 SimonGame::Initialize()안에서 수행된다.

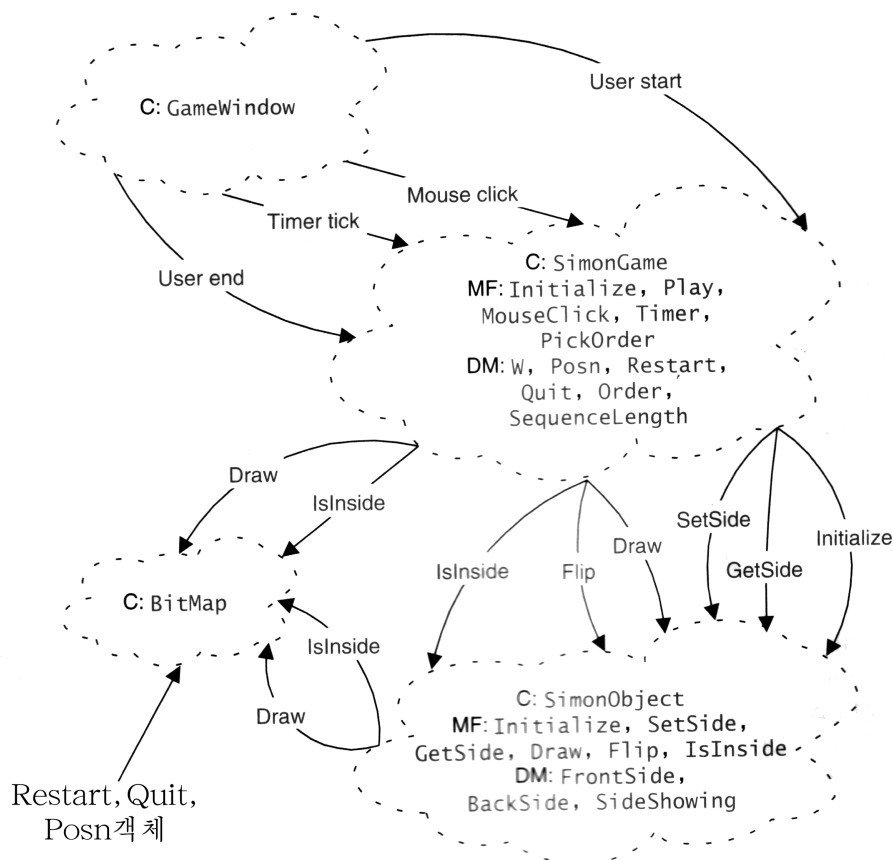



그림 10-16. Simon클라스들과 그것들이 내보낸 통보문들

```
    assert(W, GetStatus() == WindowOpen);
    SequenceLength = BeginningSequenceLength;
    X = InitialPosition.GetXDistance(MaxPositions);
    Y = InitialpositionGetYDistance(MaxPositions);
    //BitMapFile은 비트매프파일이름을 담고 있다
    vector<string> BitmapFile(MaxPositions);
    BitMapFile[0] = "c1.bmp";
    BitMapFile[1] = "c2.bmp";
    BitMapfile[2] = "c3.bmp";
    BitMapFile[4] = "c4.bmp";
  for (p=0; p< MaxPositions; ++p) {
        Posn[p] .Initialize(W, BitMapFile[p],
          "cardbk1.bmp", Position(x, y));
        Posn[p].Draw();
        X += Posn[p].GetWinth() + 0.5;
    }
```

BitMapFile은 기판상에 매 위치를 위하여 적재하려는 화상들의 화상이름들이다. for순환은 Simonobject::Initialize()를 호출하고 창문과 SimonObject의 앞면그림과 뒤면그림 그리고 SimonObject의 위치를 파라메터로 넘겨 위치를 초기화한다. Bitmap의 너비에 따라 다음 SimonObject의 위치가 계산된다.

SimonObject::Initialize()의 다음부분에서는 조종단추를 설정한다. 단순히 단추의 그림인 비트매프를 적재하고 적당한 위치에 그리기한다. 목록 10-9는 SimonObject::Initlize()의 완성된 구체례이다.

목록 10-9. SimonGame의 Initialize()성원함수

```
//initialize(): 프로그람을 초기화한다.
//이 프로그람의 객체들을 초기화한다.
void SimonGame::Initialize() {
    int p;
    float x, y;
    InitializeSeed();
    assert(W, GetStatus() = = WindowOpen);
    SequenceLength = BeginningSequenceLength;
    X = InitialPosition.GetXDistance();
    Y = InitialPosition.GetYDistance();
    //비트매프를 포함하고 있는 파일을 연다.
    vector<string> BitMapFile(MaxPositions);
    BitMapFile[0] = "c1.bmp";
    BitMapFile[1] = "c2.bmp";
```

```
    BitMapFile[2] = "c3.bmp";
    BitMapFile[3] = "c4.bmp";
    for ( p = 0; p < MaxPositions; ++p) {
        Posn[p].Initialize(W, BitMapFile[p],
          "cardbk1.bmp", Position(x, y));
        Posn[p].Draw();
        X += Posn[p].GetWindow() + 0.5;
    }
    //조종단추를 설정한다.
    Restart.SetWindow(W);
    Restart.Load("rbutton2.bmp");
    Assert(Restart.GetStatus() = = BitMapOkay);
    X = InitialPosition.GetDistance();
    Y += Posn[0].GetHeight() + 2.0;
    Restart.SetPosition(Position(x, y));
    Quit.SetWindow(w);
    Quit.Load("qbutton2.bmp");
    Assert(Quit.GetStatus() = = BitMapOkay);
    X = Restart.GetWidth(0 + 2,0;
    Quit.SetPosition(Position(x, y));
    Restart.Draw();
    Quit.Draw();
}
```

SimonObject의 Initialize()성원함수는 BitMap성원함수를 리용하여 FrontSize와 BackSide를 초기화한다. 또한 SideShowing성원함수를 Back로 설정한다. 이 함수의 코드는 목록 10-10에 있다.

목록 10-10. Simobj.cpp의 initialize()성원함수

```
//initialize(): 카드정면과 뒤면을 적재한다.
//Simon객체
void SimonObject::Initialize(Simplewindow &GameWindow,
    const string &FrontFile,  const string &BackFile,
    const Position &Posn) {
    FrontSide. SetWindow(GameWindow);
    FrontSide. SetPosition(Posn);
    FrontSide. Load(FrontFile);
    Assert(FrontSide.GetStatus() == BitMapOkay);
    BackSide.SetWindow(GameWindow);
    BackSide.SetPosition(Posn);
```

```
    BackSide.Load(BackFile);
    Assert(BackSide.GetStatus() = = BitMapOkay);
    SetSide(Back);
}
```

이 함수를 실현하면서 유희창문을 현시하는 ApiMain()코드를 작성할수 있다. 여느때에는 표시를 위하여 SimpleWindow를 전역으로 선언한다. 또한 SimonGame을 구체례화한다. 이 두 정의는 다음과 같다.

```
SimpleWindow GameWindow("SimonGame", 14.0, 7.0,
  Position(0.25,   0.25));
SimonGame Simon(GameWindow);
```

유희창문을 현시하는 ApiMain()의 예비판번호는 다음과 같다.

```
int ApiMain() {
    GameWindow.Open();
    Simon.Initialize();
    return 0;
}
```

그림 10-17 은 결과창문을 보여 준다.

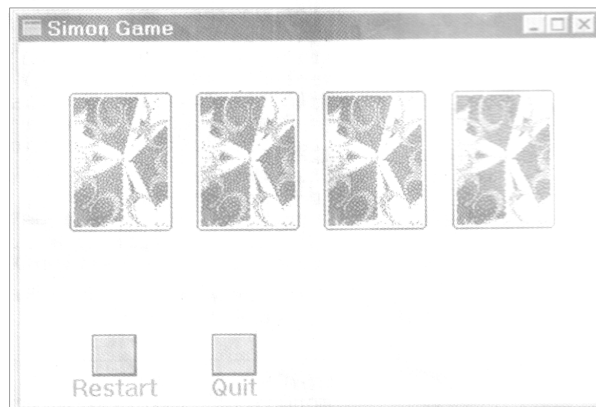



그림 10-17. 유희의 초기창문

유희를 계속하기전에 SimonObject의 구체례를 앞서 완성하자. 유희를 실행하는 기능이 필요하다. 검토자 GetSide()와 변이자 SetSide()는 우리가 작성한 검토자들과 변이자들의 기능을 가지고 있다. 이러한 기능을 목록 10-11에 보여 주었다. SimonObject의 Draw()함수는 다음과 같다.

```
void SimonObject::Draw() {
    if (SideShowing = = Back)
        BackSide.Draw();
    else
        FrontSide.Draw();
}
```

이 성원함수는 BitMap, BackSide, FrontSide를 간단히 결정한다. 또한 SideShowing의 값에 기초하여 그림을 그린다. Flip()는 Draw()를 인용한 다음 현재것의 반대켠에 SideShowing을 간단히 설정한다. 성원함수 Flip()의 정의는 다음과 같다.

//Flip():객체를 가볍게 다치고 그것을 다시 그리기한다.

```
void SimonObject::Flip() {
    SetSide(GetSide() = = Back ? Front : Back);
    Draw();
}
```

마지막으로 SimonObject::IsInside()는 BackSide.IsInside()를 호출한다. BackSide를 리용하는가 FrontSide를 사용하는가 하는것은 문제로 되지 않는다. 왜냐하면 그것들은 다 같은 위치에 있기때문이다. EzWindows는 많은 잠재적인 능력을 가지고 있다. 그러므로 SimonObject의 모든 성원함수들은 간단명료하다.

유희판을 설정하고 SimonObject를 설정하고 SimonObject를 실행하면 유희를 할수 있다 유희는 Play()성원함수에 의해 조종된다. 이 성원함수는 유희의 매 회전마다 호출된다. 매 회전은 2개의 부분으로 이루어 진다. 첫 부분은 콤퓨터가 우연적인 순서로 SimonObject를 배렬하는것이고 사용자가 같은 순서로 객체를 설정하는것이다. Simon과 사용자가 이러한 동작을 수행한다고 볼수 있다.

Play()의 첫 단계는 Simon을 변화시키고 모든 객체가 다 번져 졌는가, 번질 준비가 되였는가를 확인하는것이다. 일부 객체들이 의심할바 없이 이전 회전에서 번져 졌으므로 이러한 단계가 필요하다. 코드는 다음과 같다.

```
WhoseTurn = simon;
for (int p = 0; p < MaxPositions; ++p) {
    Posn[p].SetSide(Back);
    Posn[p].Draw();
}
```

객체의 배렬을 위한 순서를 만들수 있다. 그 순서는 성원함수 PickOrder()에 의해 진행된다. 객체를 번쩍거리게 하는 우연순서는 Simon유희자료성원인 Order에 의해 얻어 진다. 이 묶음의 원소는 Posn을 설정하는 옹근수이다. 실례로 그림 10-18에서 묶음 Order는 1, 0, 2, 3, 3값을 포함한다.

목록 10-11. Simobj.cpp모듈

```
#include <iostream>
#include <string>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "simobj.h"
using namespace std;
SimonObject::SimonObject() {
```

```
};
//Initialize():카드정면과 뒤면을 적재한다.
//SimonObject
void SimonObject::Initialize(SimpleWindow &GameWindow,
  string FrontFile, string BackFile, const Position &Posn) {
    FrontSide. SetWindow(GameWindow);
    FrontSide. SetPosition(Posn);
    FrontSide. Load(FrontFile);
    Assert(FrontSide.GetStatus() == BitMapOkay);
    BackSide.SetWindow(GameWindow);
    BackSide.SetPosition(Posn);
    BackSide.Load(BackFile);
    Assert(BackSide.GetStatus() = = BitMapOkay);
    SetSide(Back);
}
//SetSide():측면에 객체 S를 설정한다.
void SimonObject::SetSide(const Side &s) {
    SideShowing = s;
}
//GetSide():측면에 객체 S를 얻는다.
side SimonObject::GetSide() const {
    return SideShowing;
}
//GetHight():BitMap의 높이를 얻는다.
float SimonObject::GetHeight() const {
    return FrontSide.GetHeight();
}
//GetWidth():BitMap의 너비를 얻는다.
float SimonObject::Flip() {
    SetSide(GetSide() = = Back ? Front : Back);
    Draw();
}
// Flip():객체를 다치고 그것을 다시 그린다.
void SimonObject::Flip() {
    SetSide(GetSide() == Back ? Front : Back);
    Draw();
}
// IsInside():마우스가 화상을 누르기하면 결정된다.
bool SimonObject::IsInside(const Position &p) const {
```

```
        return BackSide.IsInside(p);
    }
    // 창문에 객체를 그린다.
    void SimonObject::Draw() {
        if (SideShowing == Back)
            BackSide.Draw();
        else
            FrontSide.Draw();
    }
```

이 묶음은 Posn[1]에서 SimonObject가 제일 먼저 번쩍거린다는것을 가리키며 그다음 Posn[0], Posn[2] 등의 순서로 번쩍거린다는것을 가리킨다. Posn에서 번쩍거리는 객체를 유지하는 순서를 결정하기 위해 Order에서 값을 뒤섞을수 있다.

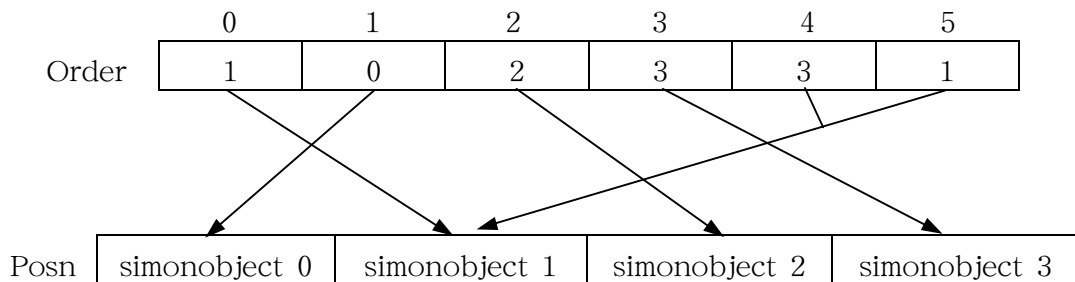



그림 10-18. 우연순서로 객체를 번쩍거리게 설정하기

Order의 뒤섞기는 성원함수 PickOrder()로 진행할수 있다. 첫 단계는 Order를 초기화하는것이다. 순서를 발생하는데 리용되는 벡토르의 원소만을 초기화해야 한다. 순서의 길이는 자료성원 SequenceLength에서 유지된다. 다음의 코드는 묶음을 초기화한다.

```
for( i = 0; i < SequenceLength; ++i)
    Order[i] = i % MaxPositions;
```

Order가 Posn보다 더 많은 원소를 가지고 있기때문에 I의 모듈연산수를 구한다. 이 단계는 Order에 보관된 값이 0부터 3까지라는것을 확인한다. 묶음을 초기화한 다음 매 회전에 대한 값을 번쩍거릴수 있다. Order의 i번째 원소에서 란수 j를 발생하며 범위는 0부터 MaxPosition-1사이이다. i번째 원소를 Order의 j번째 원소와 교체한다. 목록 10-12는 SimonGame::PickOrder의 정의를 보여 준다.

목록 10-12. Simon.cpp에서 PickOrder()성원함수

```
//PickOrder(): 우연순서를 발생한다.
void SimonGame:: PickOrder() {
    for(int i = 0; i<SequenceLength; ++i)
        Order[i] = i % MaxPositions;
    Randomint R(0, MaxPositions - 1);
```

```
        R.Randomize();
        for(int i=0; i< SequenceLength; ++i) {
            int j = R,Draw();
            int tmp = Order[i];
            Order[j] = Order[i];
            Order[i] = tmp;
        }
}
```

Play()가 순서를 발생하기 위하여 PickOrder를 호출한 다음 카드가 번쩍거릴 준비가 된다. 사건구동형객체지향프로그람에서 이러한 기능을 수행할수 있다. 그 어떤 사건도 없이 이 프로그람을 동작시킨다는것은 불가능한 일이다. SimonObject를 다치기 하는 EzWindows의 시간계수기사건을 리용해야 한다. SimonObject를 다치기 위하여 그이전의 상태를 추적해야 한다. SimonGame의 비공개부분에 성원함수 Selection을 추가한다. 그 선언은 다음과 같다.

```
int Selection;
```

Selection()은 사용자가 카드를 얻을 때마다 그 곳에서 순서적인 흔적을 유지하도록 유희의 다음단계에서 사용되게 된다. SimonGame::Play()의 마지막단계는 Selection을 0과 시작시간으로 설정하는것이다. 목록 10-13은 Play()의 완성된 코드를 보여 준다.

목록 10-13. Simon.cpp에서 Play()성원함수

```
//Play(): 현재준위에서 Simon유희를 시작한다.
void SimonGame::play() {
    //그것은 순서대로 객체를 번쩍거리게 하는 Simon의 변화이다.
    WhoseTurn = Simon;
    //번쩍거리도록 모든 객체를 변화시킨다.
    for (int p = 0; p< MaxPositions; ++p) {
        posn[p]. SetSide(Back);
        Posn[p].Draw();
    }
    //Simon객체를 번쩍거리게 하는 우연순서를 발생한다.
    PickOrder();
    //번쩍거리는 객체의 추적하기 위한 수자를 설정한다.
    Selection = 0;
    //Simon객체를 번쩍거리게 하는 시계를 동작한다.
    W.StartTimer(Flashinterval);
}
```

GameWindow의 시간계수기사건은 ApiMain()에서 설정된다. 이 호출은 다음과 같다.

```
GameWindow.SetTimerCallBack(TimerEvent);
```

성원함수 TimerEvent는 Simon의 Timer()성원함수를 호출한다. TimerEvent()의 실행은 다음과 같은 코드로 실현한다.

```
//TimerEvent(): 적당한 카드를 다치도록 유희의 시간계수기를 입력한다.
int TimerEvent() {
    return Simon.Timer();
}
```

객체가 다쳐 지는 두배의 속도로 호출하는것을 발생하는 시간계수기사건을 설정하여 SimonObjec를 숨길수 있다. SimonGame::Timer()가 처음시계박자통보를 접수하면 그것은 정면에 순서대로 첫 객체를 다치기한다. 다음통보를 접수하면 같은 객체가 뒤면을 다치기한다. 그다음 시계사건이 발생하면 다음 객체가 정면을 다치기하는 식으로 이러한 동작이 계속된다.

마지막순서의 객체의 뒤면이 다쳐 지게 된다면 시계사건은 변하게 되며 사용자의 전환이 일어 나는 표시가 나타난다. 자료성원 Selection은 묶음 Order로부터 값을 설정하기 위해 리용된다. 목록 10-14는 이와 같은 동작을 실현하는 코드이다.

Simon의 개발에서 최종단계는 마우스누르기사건을 처리하는것이다. 사용자들은 마우스를 리용하여 그것들이 다치기되여 기억한 순서대로 카드를 설정한다. 사용자는 유희를 탈퇴하고 다시 시작하기 위해 마우스를 리용할수 있다.

목록 10-14. Simon.cpp에서 Timer()성원함수

```
//Timer():시계박자사건을 처리하며 simonObject를 다치기한다.
int SimonGame :: Timer() {
    //객체를 번쩍거리게 했는가를 감시한다.
    if (Selection = = SequenceLength) {
    WhoseTurn = Player;
        W.StopTimer();
        Selection = 0;
        return 1;
    }
    //현재객체를 얻는다.
    int p = Order[Selection];
    if (Posn[p].GetSide() == Back)
        Posn[p].Flip();
    else {
        Posn[p].Flip();
        ++Selection;
    }
    return 1;
}
```

마우스누르기사건처리를 설정하는것은 시계사건을 설정하는것과 류사하다. 마우스가 그 내부에서 눌리워 졌을 때 함수 MouseClickEvent()는 마우스누르기사건을 호출하도록 GameWindow에 통보를 보낸다.

```
GameWindow.SetMouseClickCallback(MouseClickEvent)
```

MouseClickEvent()함수는 Simon의 MouseClick()성원함수에 통보를 전송한다. MouseClickEvent()의 코드서술은 다음과 같다.

```
int MouseClickEvent(const Position &MousePosn) {
    return Simon,MouseClick(MousePosn);
}
```

MouseClick()는 먼저 Restart 혹은 Quit가 설정되였는가를 검사한다. Restart단추가 설정되였다면 프로그람의 순서길이의 시작으로 Sequence를 설정하고 Play()를 호출하여 새 유희를 시작한다. Quit가 눌리면 프로그람은 완료된다. MouseClick()함수의 코드서술은 다음과 같다.

```
int SimonGame::MouseClick (const Position &MousePosn) {
    int p;
    if (Restart.IsInside(MousePosn)) {
        SequenceLength = BeginningSequenceLength;
        Play();
        return 1;
    }
    else if (Quit.IsInside(MousePosn))
        Terminate();
```

Terminate()는 응용프로그람을 완료하는 동작을 수행하는 EzWindow함수이다. 프로그람을 완료하기전에 EzWindow는 응용프로그람에 ApiEnd()통보를 보내여 필요한 해제동작을 수행하게 한다.

마우스가 조종단추의 내부를 누르지 않았다면 다음단계는 사용자의 차례인가를 결정하며 눌리워 졌다면 카드내부를 누르기하였는가를 결정한다. 마우스가 카드의 내부를 눌렀으면 정확한 순서로 카드가 설정되였는가를 검사한다. 자료성원 Selection은 묶음 Order를 다시 리용한다. Simon::Timer()에서 모든 객체가 다치기되면 Selection은 0으로 설정된다. 마우스유표가 카드의 내부에 있을 때 마우스가 눌리워 져 있는가를 결정하기 위해 Simon클라스에 Find()성원함수를 추가한다. Find()함수는 마우스가 Simon Object의 내부에서 눌리웠는가를 결정한다. Find()함수의 실행코드는 다음과 같다.

```
//Find():설정된 Simon 객체를 찾는다.
int SimonGame::fine(const Position &MousePosn) const {
    if (int p = 0; p<MaxPositions ; ++p)
        if (posn[p]; Osomsode <mousePosn))
            return p;
    return -1;
}
```

Simon::Find()의 코드는 매 Simon객체에서 성원함수 IsInSide()를 인용하여 Posn을 순환시킨다. IsInside()가 참을 되돌리면 첨수가 되돌려 진다. 순환이 진행되면 마우스는 Simon객체를 지적하지 않

으며 -1이 되돌려 진다. Find()함수를 추가하여 Simon유희의 클라스개발을 완성할수 있다. 목록 10-15는 프로그람의 마지막부분이다.

목록 10-15. simon.h에서 SimonGame의 선언

```
#ifndef SIMON _H
#DEFINE SIMON_H
#include "simobj.h"
const int MaxPositions = 4;
const int BeginningSequenceLength = 3;
const int MaxSequenceLength = 6;
const int Flashinterval = 800;
const Position InitialPosition(1.0, 1.0);
class SimonGame {
    enum Turn { Simon,  Player };
    public:
        SimonGame(SimpleWindow &Window);
        void Initialize();
        void Play();
        int Refresh();
        int MouseClick(const Position &MousePosn);
        int Timer();
        int Find(const Position & MousePosn):  const
        void PickOrder();
    private:
        SimpleWindow &w;
        Vector <SimonObject> Posn;
        BitMap Restart;
        BitMap Quit;
        Vector<int> Order;
        int SequenceLength;
        Turn WhoseTurn;
        int Selection;
};
#endif
```

Find()를 리용하여 MouseClick()를 실행할수 있다. 함수의 마지막부분은 다음과 같다.

```
else if (WhoseTurn = = Player) {
    if ((p = Find(MousePosn)) >= 0) {
```

```
                Posn[p].SetSide(Front);
                Posn[p].Draw();
                if (p != Order[Selection]) {
                    WhoseTurn = Simon;
                    W.Message("Wrong Order!");
                }
                else if (Selection + 1= = SequenceLength) {
                    if (SequenceLength ==  MaxSequenceLength) {
                        WhoseTurn = Simon;
                        W.Message("You Win!!!");
                    }
                    else {
                        ++SequenceLength;
                        play();
                    }
                }
                else
                    ++Selection;
            }
        }
        return 1;
    }
```

만일 사용자의 차례가 아니라면 마우스누르기를 무시한다. 사용자의 차례라면 마우스가 SimonObject를 지적하는가를 알아 보기 위해 Find()가 호출된다. Simon객체를 지적하지 않는다면 마우스누르기는 무시된다. 마우스가 Simon객체를 지적한다면 순서대로 놓인 다음 카드인가를 검사하고 그것을 다치기한다. 사용자가 다음 SimonObject를 정확히 설정하지 못했다면 오유통보를 내보내고 유희를 재설정한다.이때 사용자는 Restart조종단추를 눌러서 유희를 다시 시작할수 있다. 다음 SimonObject가 정확히 설정되였다면 코드는 마지막순서가 되였는가를 검사한다. 마지막순서가 아니라면 Selection은 다음순서의 객체로 증가된다. 만일 마지막객체가 설정되였다면 코드는 유희를 이겼는가를 결정한다. 만일 마지막순서라면 성공하였다는 통보가 현시되며(그림 10-19) Simon유희를 다시 시작할수 있다.

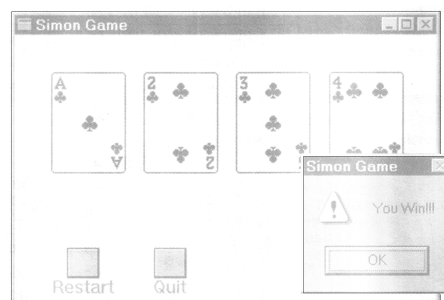



그림 10-19. 사용자가 경기에서 이겼을 때의 창문상태

Simon의 작성이 완성되였다. 목록 10-16은 SimMain.cpp에서 완성된 코드를 보여 준다.

목록 10-16 . SimMain.cpp모듈

```
#include <iostream>
#include <string>
#include <assert.h>
#include <stdio.h>
#include  "simon.h"
using namespace std;
Simplewindow GameWindow("SimonGame",
  11.5, 7.0, Position (0.15, 0.15));
int RefreshEvent() {
    return Simon.Refresh();
}
int MouseClickEvent(const Position &MousePosn) {
    return Simon.MouseClick(MousePosn);
}
int TimerEvent() {
    return Simon.Timer();
}
    int ApiMain() {
    GameWindow.Open();
    GameWindow.SetMouseClickCallbac(MouseClickEvent);
    GameWindow.SetRefreshCallback(RefreshEvent);
    GameWindow.SetTimerCallback(TimerEvent);
    Simon.Initialize();
    return 0;
}
int ApiEnd() {
    GameWindow. Close();
    return 0;
}
```

콤퓨터의 력사

Apple개인용콤퓨터

우리는 Macintosh콤퓨터를 산생시킨 Apple콤퓨터에 대해서 잘 알고 있다. 애플은 씰리콘밸리의 콤퓨터애호가들인 스테픈 워즈내크와 스테븐 죠브스에 의하여 창설되였다. 그들은 콤퓨터와 전자공학에 흥미를

가진것으로 하여 친구가 되였다. 워즈내크는 설계와 무역활동을 론쟁하며 콤퓨터와 관련된 일들에 대하여 론쟁을 벌리기 위해 모인 콤퓨터애호가들의 집단인 씰리콘밸리의 험브류콤퓨터구락부의 한 성원이였다. 그는 단기판콤퓨터를 설계제작하였다. 그 콤퓨터는 구락부를 놀래웠으며 죠브스는 한대에 500딸라로 100대를 사도록 지방콤퓨터점과 계약하였다. 그들은 애플콤퓨터회사를 창설하여 일을 시작하였다.

후에 그들은 Apple I이라고 하는 기판을 약 175개 팔았다. Apple I의 완성은 더 효과적인 콤퓨터를 창안하게끔 하였다. 그러나 그들은 방조와 자금이 더 필요하다는것을 알았다. 그들은 32살에 인텔회사에서 100만장자로 퇴직한 마이크 마쿨러기사를 우연히 알게 되였다. 그는 그들을 만나고 AppleⅡ계획에 대하여 깊은 감동을 받았으며 자금을 대주기로 하였다.

AppleⅡ는 1977년 서해안콤퓨터시장에 나왔다. 첫해에 그것은 70만딸라로 팔렸다. 4년후에는 그의 루계액이 3억3천500만딸라에 이르렀다. 1980년대에는 대중화되였으며 주식공개가격은 22딸라였다. 마지막에는 29딸라에까지 이르렀다. 죠브스와 워즈내크, 마쿨러는 순간에 100만장자가 되였다.

그림 10-20. Apple I콤퓨터를 들고 있는 스테븐 죠브스와 스테픈 워즈내크

10.9 알아 둘 점

✓ 모든것을 새로 만들려고 하지 말아야 한다. 서고코드와 API를 리용하여 응용프로그람을 개발하시오.
✓ 도형대면부를 리용하는 응용프로그람은 본문기반대면부보다 리용하기 쉽지만 개발하기 힘들다.
✓ 사건구동형프로그람은 프로그람외부에서 발생한 사건을 처리한다. 표준적인 사건으로서는 마우스누르기, 시간계수기사건, 조작체계의 통보와 같은것을 들수 있다.
✓ 속박통은 화면에서 객체의 위치를 지적하기 위한것이다. 대부분의 창문체계에서 속박통은 객체를 둘러 싸고 있는 4각형의 왼쪽웃구석과 오른쪽아래구석의 자리표에 의해 구성된다.

련습문제

10.1 창문에 번쩍거리는 통보를 내보내는 EzWindows프로그람을 작성하시오. 통보본문은 cin에 의해 입력된다. 통보는 2초에 한번씩 번쩍거린다.

10.2 인터네트에 접속하고 여러가지 응용프로그람을 개발하는데 필요한 API목록을 창조하시오. 이 목록에 API이름을 달아 주시오.

10.3 창문의 크기보다 더 큰 비트매프를 현시하는 EzWindows프로그람을 작성하시오. 어떤 현상이 나타나는가?

10.4 창문을 열도록 하는 EzWindows프로그람을 작성하시오. 마우스가 이 창문에서 눌리워 질 때 마우스자리표가 본문조종창문에 현시된다.

10.5 비트매프를 현시하는 EzWindows프로그람을 작성하시오. 이 프로그람은 BitMap파일의 이름을 사용자로부터 입력 받는다. 프로그람은 창문의 중심에 비트매프를 놓이게 한다.

10.6 임의의것을 상기시키는 프로그람을 작성하시오. 프로그람은 입력재촉에서 지정된 시간을 입력한다. 이 지정시간이 경과되면 프로그람은 적당한 차림표를 현시하여 요구되는 동작을 수행하게 한다.

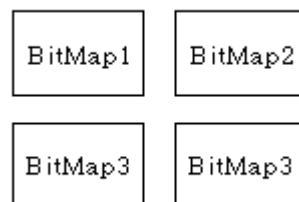
10.7 2개의 창문을 가진 프로그람을 작성하시오. 첫 창문에는 2개의 단추가 있다. 단추 한개는 투입단추이고 다른 하나는 차단단추이다. 두번째 창문에는 통보창문이 있다. 투입단추가 눌리우면 2초에 한번씩 창문에 통보가 나타난다. 차단단추가 눌리우면 통보가 껌벅거리지 않는다. 객체지향을 리용하여 설계하시오. 이것들에 대한 객체와 클라스를 개발하시오 .

10.8 Simon에서 리용한 탈퇴와 재시동단추는 전문용이 아니다. 새로운 2진화상을 리용하는 프로그람을 변경하고 더 잘 보이는 2개의 비트매프를 찾으시오.

10.9 Simon은 시작된 유희를 림시로 정지시키는 기능이였다. 이 유희에 림시정지단추를 추가하시오

10.10 Simon의 현재판본에서 제기되는 문제는 카드가 순서적으로 2번 반복되면 사용자의 귀환을 적용할수 없게 된것이다. 이미 다치기가 되기때문에 사람이 정확한 코드를 설정하였다는것을 가리키는 귀환을 접수하지 않는다. 순서가 중복된 카드를 포함할 때 동작하는 사용자귀환을 주는 방법을 설계하시오.

10.11 Simon프로그람을 변경하여 프로그람을 조금 더 높은 수준으로 힘들게 수행할수 있게 하시오. 이 유희에서 사용자가 한 유희를 성과적으로 수행하면 다음회전에서 카드가 더 빨리 눌리우게 된다. 새로운 유희는 세개의 준위가 있다. 즉 천천히, 보통, 빨리라는 준위이다.

10.12 4개의 서로 다른 비트매프를 리용하도록 프로그람을 변경하시오. BitMap는 파일(귤, 사과, 레몬, 벗), 도형(4각형, 3각형, 원, 타원), 악기(북, 트럼베트, 바이올린, 류바)와 같은것이다. BitMap는 아래와 같이 묶어 있어야 한다.

10.13 EzWindows API를 리용하여 수자식시계를 만드시오. 이 시계는 시작, 정지, 재설정단추를 가지고 있다. 이 시계는 분과 초를 현시한다(mm:ss). 이 동작을 수행하기 위하여 EzWindows API와 함께 제공되는 수자와 두점에 대한 비트매프를 리용하시오. 시간은 1초에 한번씩 변한다.

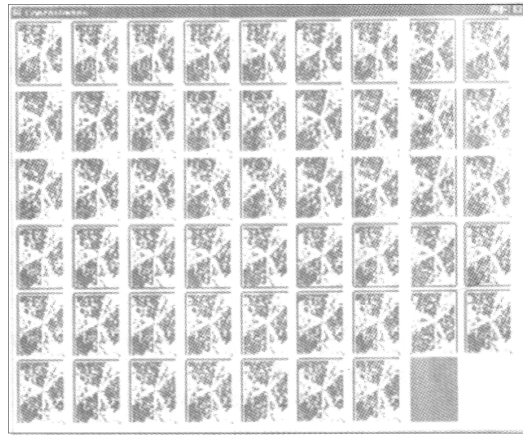
10.14 련습 10.13의 프로그람을 변화시키고 1/10초를 현시하는 시계를 만드시오. 이 시계는 1/10초에 한번씩 변한다.

10.15 5개의 카드의 부지깽이손을 현시하는 프로그람을 작성하시오, 손이 보이지 않지만 주어 진 공간을 최소화하는 방법으로 카드를 현시해야 한다.

10.16 화면보호방식을 모의하는 프로그람을 작성하시오. 이 프로그람은 2~6초간격의 우연적인 시간에 맞추어 화면의 임의의 위치에 임의의 크기의 창문을 열기도 하고 닫기도 하는 동작을 수행한다.

10.17 화면보호방식을 모의하는 프로그람을 작성하시오. 이 프로그람은 창문의 임의의 위치에 사용자가 지정한 비트매프를 여는 기능을 수행한다. 2~6초의 시간간격으로 새로운 위치에 비트매프를 그린다.

10.18 카드유희는 흔히 4각형문양을 가진 카드를 리용한다. 아래의 그림은 유희가 시작되였을 때의 창문이다. 유희를 시작하려면 2개의 카드를 서로 변화시켜서 할수 있다. 카드문양이 같다면 1개 점을 얻는다. 그렇지 않으면 카드는 뛰여 넘게 된다. 사용자가 같은 문양의 카드를 뛰여 넘는다면 다른 변화가 일어 난다. 같은 문양을 맞추지 못한다면 다른 사용자에게 조종권이 넘어 간다.

CD-ROM에서 제공된 카드화상을 리용하여 이러한 유희를 하는 프로그람을 작성하시오.

10.19 련습 10.18의 유희프로그람을 작성하시오. 콤퓨터에 의해 변화된 카드는 기억된다. 그러나 사용자가 변환한 카드는 기억되지 않는다. 현재의 점수를 보여 주는 창문을 창조하시오. 점수는 얼마나 많은것을 찾았는가를 반영하는 수자이다. 유희가 완료되면 콤퓨터는 첫 상태로 간다.

10.20 련습 10.19에서 작성한 유희에 첫 상태로 가도록 설정하는 단추를 포함시키시오.

10.21 련습 10.20에서 작성한 유희를 변화시켜서 콤퓨터가 제정된 시간의 60%내에 다치기하는 카드의 위치를 기억하게 하시오. 콤퓨터를 이길수 있는가? 프로그람을 변화시켜서 콤퓨터가 다치기된 카드의 80%를 재호출하도록 하시오.

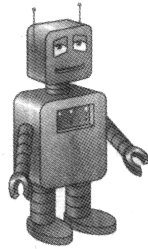
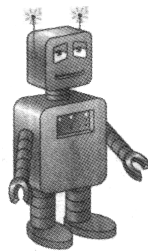
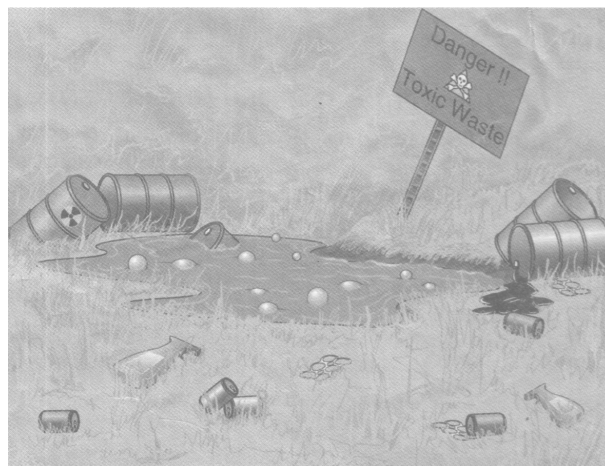
10.22 지금 에네르기분야에서는 위험한 환경속에서 리용할수 있는 로보트를 개발하고 있다. 로보트 즉 위험환경로보트관측기(HERO)는 단순한 지령에 의해 조종된다. HERO는 다음과 같은 지령을 리해한다.

지령	동작
U 혹은 u	꼭대기로 올라가기
D 혹은 d	밑으로 내려가기
L 혹은 l	왼쪽으로 이동
R 혹은 r	오른쪽으로 이동
Mn 혹은 mn	이동거리를 n미리메터로

HERO가 운동지령(U.D.L.R)을 실행하면 그 방향으로 움직인다. 로보트는 마지막설정인 운동거리지령(M)에 의하여 지정된 거리만큼 움직인다. 운동지령의 묶음은 옹근수이다. HERO를 개발하기 위해 HERO의 동작을 표시하는 프로그람을 작성하시오. 프로그람은 건반으로부터 HERO지령을 읽고 창문에 HERO동작을 표시한다. HERO의 목적이 위험한 물질을 발견하는것이기때문에 프로그람은 HERO가 위험한 물질을 가진 구역을 넘어 갈 때마다 그 구역을 화면에서 강조해 주어야 한다. HERO의 지정파일은 다음과 같은 형식으로 되여 있다.

```
20.0    20.0
9.0     9.0
0.0     0.0
M 10
D
D
D
R
L
```

L
U
U

첫행은 창문의 너비와 높이이다. 두번째 행은 창문에서 위험한 구역의 위치이다. 셋째 행은 HERO의 시작자리표이다. 그다음 3개 행은 HERO의 지령이다. 프로그람은 3개 행을 읽고 지정된 크기의 창문을 창조하고 위험지역을 그리며 시작과 표에 HERO를 그린다. 초기현시가 설정된후에 프로그람은 HERO지령을 읽고 HERO의 움직임을 보여 준다. 프로그람은 프로그람의 마지막지령이 실행될 때 완료된다. HERO를 나타내기 위하여 비트매프를 리용하시오. HERO는 위험지역의 밖에 있을 때는 다음의 BitMap로 표시된다.

BitMap의 이름은 이 책에 부속되여 있는 CD-ROM의 BitMap등록부의 robot.bmp이다. HERO가 위험지역에 들어 가면 robot.bmp가 현시된다. HERO가 창문의 왼쪽면의 경계를 넘어서 움직이려 한다면 창문의 오른쪽에 HERO가 다시 나타나게 된다.

이와 같이 HERO가 창문의 오른쪽경계면을 넘어 서면 HERO가 다시 왼쪽면에 나타나야 한다. HERO가 창문의 꼭대기 혹은 밑에 있게 되며 그때 동작도 우에서 설명한것과 같이 된다. 즉 HERO가 창문의 꼭대기면을 넘어 서게 되면 창문바닥에서 HERO가 다시 나타나며 HERO가 창문바닥면을 넘어 서게 되면 창문꼭대기에서 다시 나타나게 된다. 위험구역을 표시하기 위하여 BitMap hazard.bmp를 리용한다. BitMap는 다음과 같은 그림이다.

제 11 장. 지적자와 동적기억기

소 개

많은 문제풀이들에서는 정보량들을 표시할것을 요구한다. 이러한 과제들은 실행시 객체들의 동적창조와 관리를 제공해 주는 C++기구에 의하여 실현될수 있다. 이 기구는 프로그람을 유연하게 하며 임의의 크기를 가진 자료들을 표시하게 한다. 동적객체들은 new연산자를 리용하여 창조되며 delete연산자를 리용하여 체계에 돌려 진다. 동적객체는 지적자를 통하여 접근되는데 여기서 지적자(pointer)란 그의 값이 다른 객체의 위치로 되는 객체이다. 지적자는 반복자(iterator)와 비슷하다. 사실 반복자는 지적자추상화(pointer abstraction)로서 나타난다. C++는 지적자들을 지원하기 위하여 2개의 보충연산자인 주소연산자 &와 참조해제연산자 *를 제공한다. 주소연산자는 객체의 위치를 계산하며 참조해제연산자는 그 위치에 보관된 값을 계산한다. 우리의 론의는 지적자로부터 시작된다.

기본개념

- 왼쪽값
- 오른쪽값
- 지적자형
- 빈 주소
- 참조해제연산자 *
- 간접성원선택연산자 ->
- 주소연산자 &
- 지적자대입
- 간접대입
- 지적자로서의 파라메터
- 지적자에 대한 지적자
- 상수지적자
- 상수에 대한 지적자
- 배렬과 지적자
- 지령행파라메터
- 함수에 대한 지적자
- 동적객체
- 빈 기억구역
- new와 delete연산자
- 례외처리
- set_new_handler()
- 예속지적자
- 기억기손실
- 해체자
- 복사구축자
- 성원값주기
- this지적자
- const수식자
- 함수에 대한 지적자
- 례외처리

11.1 왼쪽값과 오른쪽값

C++에는 두가지 종류의 식이 있는데 하나는 평가와 수정을 다 할수 있는 객체를 표시하는 식과 다른것은 평가만 할수 있는 객체를 표시하는 식이다. 실례로 다음과 같이 정의되였다고 가정하자.

```
int a = 1;
int b;
int c[3];
```

다음의 코드토막은 두 종류의 식을 포함한다.

```
b = 5;
cout << b << end1;
c[0]=2*a;
```

식 b는 때에 따라 그 값이 평가되거나 수정될수 있는 객체를 표시한다. 첫번째 값주기명령문에서 식 b로 표시된 객체는 수정된다. 삽입명령문에서 식 b로 표시된 객체는 평가된다. 두번째 값주기에서 식 c[0]은 그 값이 평가되거나 수정될수 있는 객체를 표시한다.

평가 및 수정을 다 할수 있는 객체를 표시하는 식은 왼쪽값(lvalue)이다. 즉 우의 코드에서 a, b와 같은 객체의 이름은 왼쪽값이지만 앞서 론의한것처럼 왼쪽값이 다 객체의 이름은 아니다.

전형적인 값주기명령문에서 객체이름을 왼쪽연산수로 리용하지만 값주기명령문의 문법은 왼쪽연산수가 왼쪽값이라는것만을 요구한다. 이 유연성은 객체가 다 이름을 가지는것은 아니기때문에 중요하다. 실례로 배렬이나 벡토르는 이름을 가지지만 개별적인 요소는 가지지 않으며 개별적인 요소들은 왼쪽값첨수식을 리용하여 참조된다. 그러므로 우의 코드토막에서 c[0]에 대한 값주기는 이름을 가지지 않는 객체에 대한 값주기실례이다.

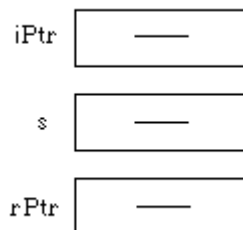
값주기명령문에서 두개의 오른쪽연산수 5와 2*a는 왼쪽값이 아니므로 5나 2*a를 값주기대상으로 한다는것은 의미를 가지지 않는다. 이런 종류의 식을 오른쪽값(rvalue)이라고 하는데 값주기명령의 왼쪽연산수로는 리용될수 없다. 왼쪽값은 이러한 제한을 받지 않는다. 왼쪽값은 값주기에서 왼쪽뿐아니라 오른쪽연산수로도 리용될수 있다.

11.2 지적자기초

지적자는 그 값이 다른 객체의 위치를 표시하는 객체이다. 지적자객체는 흔히 참조해제연산자로 알려 진 단항간접연산자(unary indirection operater) *와 결합하여 정의된다.

다음의 코드토막은 3개의 지적자객체 iPtr, s, rPtr를 선언하였다.

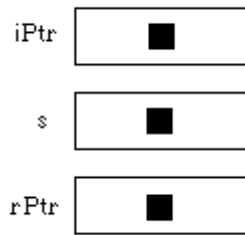
```
int *iPtr ;
char *s;
Rational *rPtr ;
```

객체 iPtr는 int형지적자이며 s는 char형지적자, rPtr는 Rational형지적자이다. int, char, Rational이 다 다른 형이므로 이러한 지적자형들은 모두 다르다. 이 선언에서는 초기화가 없으므로 3개 객체는 아직 초기화되지 않았다. 다음의 그림은 이 정의결과를 보여 준다(가로선은 초기화되지 않은 값을 가리킨다).

지적자형은 참조해제연산자와 련결된 결합형을 리용하여 참조된다. 즉 int*는 int형지적자 , char*

는 char형지적자라고 읽는다.

임의의 지적자객체에 값주기할수 있는 기호값이 하나 있다.그 기호값은 0인데 빈 주소라고 한다. 실례로 다음의 명령문은 모두 빈 주소를 값주기한것이다.

```
int *iPtr = 0;
char s = 0;
Rational rPtr = 0;
```

값이 빈 주소인 지적자객체는 호출될수 있는 객체를 지적하지 못한다. 값이 0인 지적자를 구분하기 위하여 안에 검은 칠을 한 4각형을 리용한다.

다음절에서는 개별적인 객체와 결합되여 지적자로 리용되는 연산자에 대하여 보게 된다. 그렇게 한다면 그림 11-1과 같은 결과를 얻을수 있을것이다.

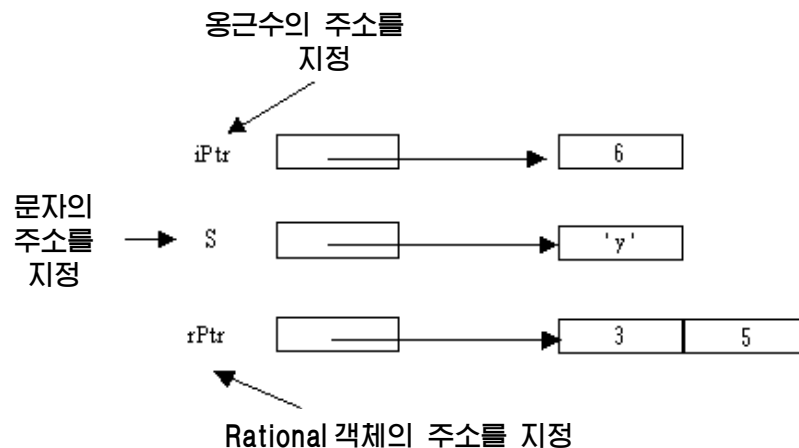



그림 11-1. 3개의 지적자객체들과 그것들이 지적하는 값들

값주기연산자는 지적자객체에서도 정의된다. Ptr1과 Ptr2가 같은 지적자형이라면 다음의 값주기가 성립한다.

```
Ptr1 = Ptr2;
```

이 값주기의 의미는 다른 객체에 대한 값주기의미와 같다. 즉 Ptr1이 표시하는 값은 Ptr2가 표시하는 값으로 수정된다. 이러한 값주기의 결과와 같이 Ptr1과 Ptr2는 같은 객체를 지적하게 된다. 우의 실례에서 지적자객체의 이름을 오른쪽연산수처럼 리용하지만 오른쪽연산수는 사실상 왼쪽연산수와 같은 형으로 평가되는 임의의 식이다.

실례로 다음과 같이 정의되였다고 가정하자.

```
int i=1;
int *iPtr;
```

char *cPtr;

다음의 3개 값주기명령문들은 성립할수 없다. 왜냐하면 오른쪽연산수들이 같은 형의 값을 지적하지 않기때문이다.

iPtr=i; //틀림: i는 int형지적자가 아니다
i=iPtr; //틀림: iPtr는 int형이 아니다.
iPtr=cPtr; //틀림: cPtr는 int형지적자가 아니다.

우의 첫 값주기명령문에서 iPtr는 값주기대상이다. 그러므로 오른쪽연산수는 int형지적자값으로 되여야 하지만 현재 i는 int형이므로 값주기가 성립하지 않는다. 두번째 명령문에서 오른쪽연산수는 int형으로 평가되여야 한다. 그런데 iPtr가 int형지적자이므로 성립하지 않는다. 세번째 값주기명령문에서 오른쪽연산수는 int형지적자로 되여야 한다. 그렇지만 cPtr는 char형지적자이므로 성립하지 않는다. 이 값주기명령문들은 오른쪽과 왼쪽값주기연산수들이 같은 형으로 되여야 C++문법에 맞는다.

11.2.1 주소화와 간접

객체의 위치가 프로그람에 의하여 계산되고 리용되게 하기 위하여 C++는 단항주소연산자(unary address operator) &를 리용한다. 주소연산자가 식에서 객체에 적용될 때 주소연산에 참가하는 값은 객체에 대한 지적자이다.

주의

하나의 명령문에 하나의 객체를 정의하여야 한다.

지적자객체를 정의할 때 어떤 프로그람작성자들은 다음과 같이 객체이름앞이 아니라 형이름 다음에 참조해제연산자를 붙여 쓴다.

char* Ptr;

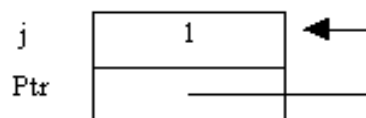
이 형태는 리용되는 형이 지적자형이라는것이 명백하므로 잘 리용된다. 그렇지만 오유로 될수도 있다. 실례로 다음의 명령문을 보기로 하자.

char* s, t; //실지는 char *s; char t이다.

명령문은 처음에 char 형지적자 객체 s를 창조한다. 다음 char형객체 t를 창조한다. 차이나는 객체형은 참조해제연산자가 정확히 결합된 결과이다. 즉 참조해제연산자가 char형과 함께 나란히 놓인다 하더라도 참조해제연산자는 객체 s와 관련된다. 이미 경고하였지만 한개 명령문에 객체를 한개씩 선언해야 한다. 오유의 이러한 형태의 무시는 경고오유발생의 원인으로 된다.

다음의 코드토막의 실행을 가정해 보시오.

int j =1;
int *Ptr;
Ptr = &j;

첫번째 정의는 j를 1로 초기화한다. 값주기명령문은 j의 기억기위치주소에 Ptrl을 설정한다. 아래에 명령문의 실행결과를 그림으로 보여 주었다.

j의 값은 j를 리용하여 직접 호출할수도 있고 Ptr지적자객체를 리용하여 간접적으로 호출할수도 있

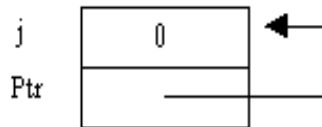
다. 간접방식은 참조해제연산자 *를 리용하여야 한다. 참조해제연산자가 지적자객체로 리용될 때 참조해제연산은 지적자객체가 가리키는 객체의 왼쪽값을 구한다. 실례로 다음의 삽입명령문에서 값 1이 표준출력흐름에 표시된다.

```
cout << *Ptr  <<end1; // 객체 j를 표시한다.
```

*Ptr는 Ptr가 지적하는 객체이므로 1이 표시된다. Ptr가 객체 j가 보관된 기억기주소를 지적하므로 *Ptr의 값은 1이다. 따라서 참조해제연산자가 지적자객체로 작용할 때 연산자의 기능은 그 연산자가 반복자에 대하여 작용할 때와 비슷하다. 실례로 반복자들은 가상지적자로 볼수 있다.

왼쪽값에 대하여 앞에서 본것처럼 값주기연산자의 왼쪽연산수는 임의의 왼쪽값이 될수 있다. 이러한 유연성은 다음의 값주기명령문에서 보여 준다.

```
*Ptr = 0; //객체 j를 수정
```

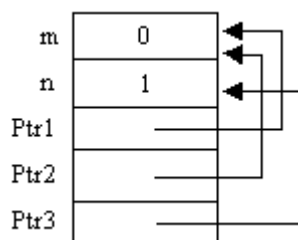
오른쪽연산수는 0이며 왼쪽연산수는 식 *Ptr이다. 식 *Ptr는 j를 가리키는 왼쪽값이다. 즉 j에 간접적으로 0을 대입한다. 그러므로 이 값주기연산은 j 에 0을 주기 위하여 간접적으로 수정한다. 객체 j와 Ptr에 대한 기억기모형은 다음과 같다.

아래의 두 출력명령문들이 실행되면 표준출력장치에 0이 두번 표시된다.

```
cout << j <<end1;
cout << *Ptr << end1;
```

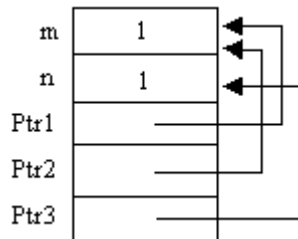
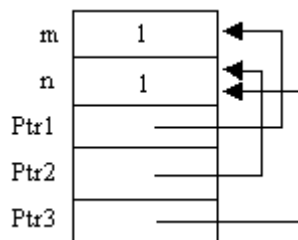
다른 객체와 마찬가지로 지적자객체의 정의는 다음의 실례에서와 같이 여러가지로 초기화식으로 될수 있다.

```
int m=0;
int n=1;
int  *Ptr1=&m;
int  *Ptr2=Ptr1;
int *Ptr3=&n;
```

처음 두개의 정의는 int형 m과 n을 창조하고 초기화한다. 그다음 세개의 정의는 int형지적자객체 Ptr1와 Ptr2, Ptr3을 창조하고 초기화한다.

지적자 Ptr1은 m를 지적하기 위하여 초기화된다. 지적자 Ptr2는 Ptr1의 현재값을 복사하기 위하여 초기화한다. 그러므로 Ptr2 역시 m을 가리킨다. 마지막으로 Ptr3은 n을 지적하기 위하여 초기화된다. 5개의 객체에 대한 기억기모형은 다음과 같다.

다음의 코드토막의 실행을 가정하여 보시오. 이 객체들에서 변환은 무엇인가?

```
*Ptr1= * Ptr3;
Ptr2= Ptr3;
```

첫번째 값주기에서 오른쪽연산수는 식 *Ptr3이다. Ptr3이 n의 주소를 가리키는 지적자이므로 식 *Ptr3은 1로 된다. 첫번째 값주기에서 왼쪽연산수는 식 *Ptr1이다. Ptr1가 m의 주소에 대한 지적자이므로 식 *Ptr1은 m의 값을 가리키는 정확한 왼쪽값이다. 그러므로 첫번째 값주기는 m의 값을 n의 복사로 하기 위하여 간접적으로 m을 수정한다. 이때 기억기모형은 다음과 같다.

코드토막의 두번째 값주기에서 Ptr2의 값은 Ptr3의 값으로 교체된다. Ptr3은 n을 가리키므로 Ptr2 역시 n을 가리킨다. 기억기모형은 다음과 같다.

만일 지적자객체가 그때 클라스형객체를 가리킨다면 참조해제연산자와 선택연산자를 리용하여 개별적성원들을 호출할수 있다. 선택연산자가 우선권이 더 높으므로 참조해제연산자는 괄호안에 넣어야 한다.

```
Rational a(4, 3);
Rational *aPtr = &a;
(*aPtr).Insert(cout); //객체 *aPtr의 Insert()성원을 호출한다. 괄호가 필요하다.
```

이 문법은 쓰기 불편하므로 C++에서는 선택과 참조해제를 결합한 간접성원선택연산자 ->를 포함한다. 따라서 다음의 명령문은 aPtr가 가리키는 Rational객체도 표시한다.

```
aPtr -> Insert(cout);
```

참조해제 및 간접성원선택연산자들은 클라스의 참조규칙을 극복하기 위한 도구가 아니다. 비공개가 아닌 자료성원은 이러한 연산자를 리용하는 성원으로 참조될수 없다. 실례로 다음의 명령문은 성립되지 않는다.

```
(*aPtr ).NumeratorValue = 1;// 틀림: private성원
aPtr -> DenominatorValue = 2; // 틀림: private성원
```

값이 빈 주소인 지적자객체에 대해서는 값을 대입할수 없다. 실례로 다음의 코드토막은 성립하지 않는다.

int *NullPtr = 0 ;
*NullPtr = 1; // 틀림: NullPtr는 int형객체의 주소를 지적하고 있지 않다.

11.2.2 지적자에 대한 지적자

C++에서는 객체가 지적자에 대한 지적자형으로 되는 지적자형도 제공한다. 또한 객체가 지적자의 지적자에 대한 지적자로 되는 지적자형도 제공한다. 개념이 새롭기는 하지만 리용되고 있다(실례로 동적인 다차원목록). 아래에 int*객체를 가리키는 지적자형 PtrPtr를 정의하였다.

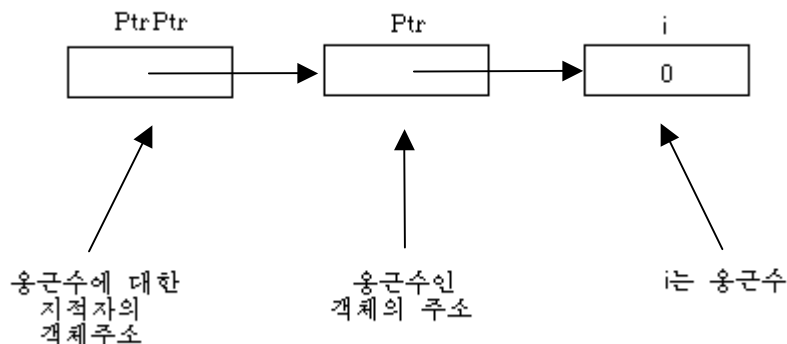
int **PtrPtr ;

정의에서 매개 *는 또 다른 참조해제준위를 가리킨다. 실례로 PtrPtr앞에 있는 두개의 *는 지적자에 대한 지적자를 의미하며 *가 3개이면 지적자의 지적자에 대한 지적자를 의미한다. 아래와 같이 정의하여 보시오.

int i = 0;
int *Ptr = &i;

Ptr가 int *이므로 다음의 값주기는 정확하며 식 &Ptr를 int **로 평가한다.

PtrPtr=&Ptr;

값주기결과와 같이 객체 i, Ptr 그리고 PtrPtr는 다음의 관계를 가진다.

앞에서 본것처럼 C++는 두개의 지적자객체가 지적하는 값의 형이 차이나면 다른 형의 지적자객체로 인정한다. 그러므로 다음의 값주기는 성립하지 않는다.

PtrPtr=Ptr; //틀림

PtrPtr는 int**를 요구한다. 오른쪽연산수는 int*이지만 PtrPtr는 int**를 요구하므로 값주기가 성립하지 않는다.

11.2.3 파라메터로서의 지적자

이때까지는 실례들을 토막토막 나누어 보았다. void형 IndirectSwap()함수를 리용하는 자그마한 프로그람을 프로그람 11-1에 주었다. IndirectSwap()함수는 두개의 형식파라메터 Ptr1과 Ptr2를 가지고 있다. 이 두 파라메터는 char*형파라메터값이다.

경험

지적자

값주기명령문에서 이름을 리용하지 않고 값을 호출하거나 변경하는것이 지적자객체를 매우 편리하게 하지만 쉽게 혼돈을 일으킨다. 지적자에 의하여 생긴 프로그람오유는 애매하며 찾아내기 힘들므로 지적자를 어떻게 리용했는지 주의깊게 살펴보아야 한다. 지적자를 쉽게 리용하는 묘리는 객체들의 어느것이 어느것을 지적하는가를 그림으로 그려 보는것이다. 이 장에서는 지적자들이 어떻게 작용하는가를 설명하는데 이러한 묘리를 리용하였다. 숙련된 프로그람작성자들까지도 지적자를 포함한 코드를 리해하고 오유를 수정하는데서 자료구조를 그림으로 그리는 방법을 리용한다.

```
// 프로그람 11-1. 간접을 리용한 객체바꾸기
#include <iostream>
#include <string>
using namespace std;
void IndirectSwap(char *Ptr1, char *Ptr2) {
    // Ptr1과 Ptr2가 지적하는 문자형객체의 내용을 교환한다.
    char c = *Ptr1;
    *Ptr1 = *Ptr2;
    *Ptr2 = c;
}
int main() {
    char a = 'y';
    char b = 'n';
    IndirectSwap(&a,&b);// IndirectSwap()에 a, b의 값을 넘긴다.
    cout <<a <<b <<endl; // a와 b의 새로운 값을 표시한다.
    return 0;
}
```

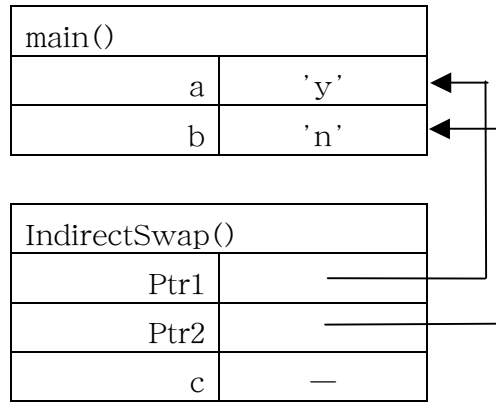
프로그람 11-1. 지적자가 참조파라메터를 모의할수 있다는것을 보여 주는 실례

프로그람이 시작되면 객체 a와 b는 'y'와 'n'으로 각각 초기화된다. 다음의 그림은 a와 b를 정의한후 main()함수의 실지기록을 보여 준다.

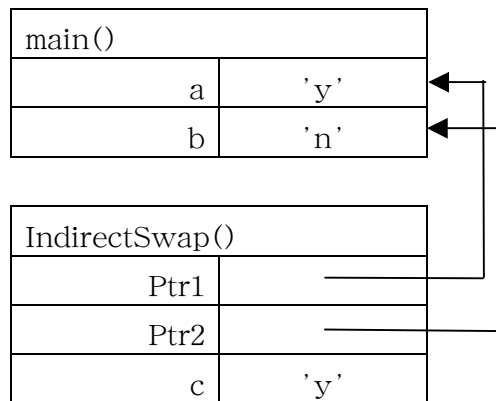
main()	
a	'y'
b	'n'

프로그람 11-1에서 IndirectSwap()함수의 호출은 IndirectSwap(&a, &b)이다. 이때 이 호출을 위한 실제파라메터는 식 &a와 &b의 값이다. 식 &a는 a를 가리키므로 값파라메터 Ptr1은 a를 지적하며 식 &b는 b를 가리키므로 값파라메터 Ptr2는 b를 지적한다. IndirectSwap()함수의 호출과 관련한 기억기모형은 아래와 같다.

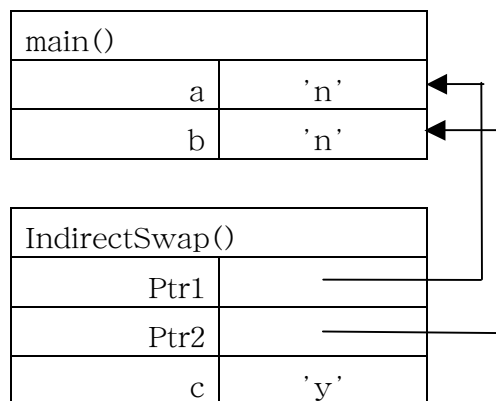
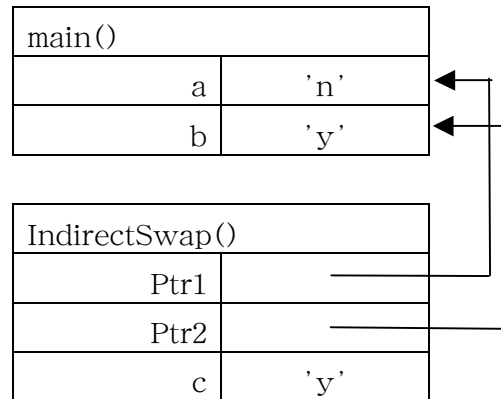
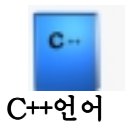
main()	
a	'y'
b	'n'

IndirectSwap()	
Ptr1	
Ptr2	
c	—

함수 IndirectSwap()의 국부객체 c는 식 *Ptr1의 값으로 초기화된다. 특히 IndirectSwap()함수를 호출할 때 객체 c는 Ptr1이 'y'를 가지는 main()함수의 객체 a를 가리키므로 'y'로 초기화된다.

main()	
a	'y'
b	'n'

IndirectSwap()	
Ptr1	
Ptr2	
c	'y'

그때 IndirectSwap()에서 두개의 값주기명령문은 Ptr1과 Ptr2가 지적하는 객체를 수정한다.

```
*Ptr1 = *Ptr2;
*Ptr2 = c;
```

Ptr1이 main()함수의 객체 a를 가리키므로 왼쪽값 *Ptr1을 수정하는데서 첫번째 값주기는 실지 a를 수정한다. 따라서 IndirectSwap()함수의 실지기록에서는 이에 해당한 값들을 주지 않는다. a의 새로운 값은 *Ptr2의 왼쪽값이다. Ptr2가 main()함수에서 객체 b를 가리키므로 이 값주기명령문은 a에 'n'라는 값을 준다.

IndirectSwap()함수의 두번째 값주기에서 왼쪽연산수는 *Ptr2이다. Ptr2가 main()함수의 객체 b를 가리키므로 실지는 b를 수정한다. 앞의 값주기와 마찬가지로 이 값주기는 IndirectSwap()함수의 실지기록에서는 아무 값이나 변경시키지 않지만 main()함수의 실지기록의 내용은 갱신된다. b의 새로운 값은 c와 같이 'y'이다.

IndirectSwap()함수의 실행은 두번째 값주기에 의하여 완성되며 조종흐름은 main()함수로 돌아 간다. 결국 IndirectSwap()의 호출은 실제파라메터가 가리키는 객체의 값을 바꾸는것이다. IndirectSwap()함수는 6장에서 본 Swap()함수와 비슷하다.

참조파라메터의 중요성

프로그람 11-1은 참조해제와 주소연산자 그리고 값파라메터를 리용하여 참조파라메터를 모의할수 있다는것을 보여 준다. 사용자가 C++프로그람작성법을 습득하였다면 이 기능은 그리 중요하지 않다. 그러나 C가 참조파라메터를 제공하지 않기때문에 C프로그람작성자들에게는 아주 중요하다. C프로그람작성자들은 참조파라메터를 가진 함수가 필요할 때마다 함수에 대한 실지파라메터가 수정해야 할 객체에 대한 지적자라는것을 확인하고 지적자조작을 수행하는 함수를 작성해야 한다. 이 방법은 쓰기 불편하고 오유가 생기기 쉽다. C++에서 참조파라메터의 도입은 쏘프트웨어기술분야에서 의의가 크다.

문 제

1. 옹근수객체 Counter를 지적하는 지적자 IntPtr를 만드는 C++명령문을 작성하시오.
2. 지적자 IntPtr1이 IntPtr2와 같은 위치를 가리키는 C++명령문을 작성하시오.
3. Ptr2가 가리키는 기억기위치에 Ptr1이 지적하는 값을 복사하는 C++명령문을 작성하시오.
4. 지적자 FloatPtr1이 가리키는 류점수객체에 값 34.5를 대입하는 C++명령문을 작성하시오.
5. Ptr1과 Ptr2는 두개의 double객체에 대한 지적자이다. Ptr1과 Ptr2의 값을 바꾸는 명령문을 작성하시오. 즉 Ptr2는 Ptr1이 가리키던 주소를, Ptr1는 Ptr2이 가리키던 주소를 가리킨다.
6. Ptr1과 Ptr2는 double객체이다. Ptr1과 Ptr2가 가리키는 주소에 있는 값을 바꾸는 명령문을 쓰시오.
7. 다음의 프로그람의 결과값을 구하시오.

```
#include <iostring.h>
    int main() {
    int Value = 10;
    int *Ptr = &Value;
```

```
        cout << *Ptr << end1;
        return 0;
    }
```

8. 다음의 선언과 명령문을 분석하시오.

```
int s[10];
int  *iPtr1;
int  *iPtr2;
int  **iPtr3;
for (int i = 0;  i <10; i ++)
    s[i] = 9-i;
iPtr1 = s;
iPtr2 = &s[3];
iPtr3 = &iPtr1;
*iPtr1 = 5;
*iPtr2 = 11;
**iPtr3 = 14;
```

다음의 그림에 우의 프로그람이 실행된후 기억기에 있는 값(오른쪽값과 왼쪽값)을 써넣으시오. 옹근수와 지적자는 4byte기억구역을 차지한다고 가정하시오.

s[0]		1000
s[1]		1004
s[2]		1008
s[3]		1012
s[4]		1016
s[5]		1020
s[6]		1024
s[7]		1028
s[8]		1032
s[9]		1036
iptr1		1040
iptr2		1044
iptr3		1048

9. 다음의 프로그람의 결과값을 구하시오.

```
#include <iostream>
int main() {
    int Value =10;
    int *Ptr = &Value ;
    *Ptr =0;
    cout << Value <<end1;
    return 0;
```

```
}
```

10. 다음의 프로그램의 결과값을 구하시오.

```
#include <iostream>
int main() {
    int Value1= 10;
    int Value2= 40;
    int *Ptr1=&Value 1;
    int *Ptr2=&Value2;
    Ptr1 = Ptr2;
    cout << *Ptr1 << end1;
    cout << *Ptr2 << end2;
    return 0;
}
```

11. 다음의 프로그램의 결과값을 구하시오.

```
#include <iostream>
void Test(int *p, int v) {
    *p = 5;
    return ;
}
int main() {
    int Count = 5;
    Test(Count, Count);
    cout << Count << endl;
    return 0;
}
```

12. 다음의 프로그램의 결과값을 구하시오.

```
#include <iostream>
void Swap(int *p1, int *p2) {
    int *temp = p1;
    p1 = p2;
    p2 = temp;
    return;
}
int main() {
    int Value1 = 10;
    int Value2 = 30;
    int *Ptr1 = &Value1;
    int *Ptr2 = &Value2;
```

```
    Swap(Ptr1, Ptr2);
    cout << Value1 << Value2 << endl;
    return 0;
}
```

11.3 상수지적자와 상수에 대한 지적자

const변경자를 지적자정의에서도 리용할수 있다. const변경자가 놓이는 위치에 따라 지적자가 상수로 되거나 지적자가 가리키는 위치의 내용이 상수로 된다. 다음과 같이 정의하여 보시오.

```
char c1='a';
char c2='b';
```

다음의 코드토막은 const변경자가 적용된 지적자정의에서 3가지 실례이다.

```
const char *Ptr1 = &c1;// *Ptr는 상수로 된다. Ptr1은 상수가 아니다.
char const *Ptr2 = &c1; // Ptr2는 상수로 된다. Ptr2는 상수가 아니다.
char *const Ptr3=*&c1; // Ptr3은 상수이다. *Ptr3은 상수가 아니다.
```

const변경자는 자료형의 앞 혹은 뒤에 쓸수 있는데 const변경자는 단항연산자 *다음에도 쓸수 있다. 언어정의를 따른다면 변경자는 자료형의 앞 혹은 뒤에 놓여도 그 의미가 달라 지지 않는다. 그러므로 앞의 코드토막에서 처음 2개의 명령문은 같은 의미를 가진 지적자객체를 정의할수 있다. 우의 실례에서 설명하였다. 실례로 Ptr1의 정의와 함께 설명문이 리용되였다. 개별적인 지적자정의와 같이 Ptr1가 지적하는 객체의 값은 *Ptr1에 값주기하여서는 변경할수 없다. 그러나 Ptr1이 지적하는 객체 c1에 직접 값주기하여 변경할수 있다.

```
*Ptr1= 'A' ; // 틀림 ; *Ptr는 상수이다.
c1= 'A' // 옳다; c1은 상수가 아니다.
```

어느것이 상수로 정의되였는가를 결정하는 방법은 정의뒤부분을 읽어야 하는데 정의에서는 *를 a에 대한 지적자라고 한다. 따라서 Ptr1에 대한 정의는 《Ptr1은 char형상수에 대한 지적자》이고 Ptr2에 대한 정의는《Ptr2는 상수 char에 대한 지적자》이며 Ptr3에 대한 정의는 《Ptr3상수는 char형에 대한 지적자》라고 할수 있다. 마지막해설이 문법적으로 좀 어색하지만 이 방법은 어느것이 상수이고 어느것이 상수가 아닌가를 쉽게 식별할수 있게 한다. 우의 정의에서 *Ptr1, *Ptr2, Ptr3은 상수이다. 그러므로 다음의 명령은 성립할수 없다.

```
*Ptr1 = 'A'; // 틀림; *Ptr1은 변경할수 없다.
*Ptr2 = 'B' ; // 틀림; *Ptr2는 변경할수 없다.
Ptr3 = Ptr2; // 틀림; Ptr3은 변경할수 없다.
Ptr1, Ptr2, *Ptr3은 상수가 아니므로 아래와 같이 명령문을 쓸수 있다.
Ptr1 = Ptr2; // 옳다; Ptr1은 변경할수 있다.
Ptr2 = Ptr1; // 옳다; Ptr2는 변경할수 있다.
*Ptr3 = 'C'; // 옳다; *Ptr3은 변경할수 있다.
```

아래에 내용이 상수인 위치에 대한 상수형지적자를 정의한다.

const char *const Ptr4 = &c1;

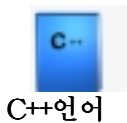
그러므로 다음의 값주기명령문은 성립할수 없다.

Ptr4 = &c2;// 틀림; Ptr는 변할수 없다.
*Ptr4 = 'd' // 틀림; *Ptr는 변할수 없다.

첫번째 명령문은 Ptr4를 변경시키려고 하였기때문에 오유로 된다. 두번째 명령문은 Ptr4가 가리키는 위치의 내용을 변경시키려고 하였기때문에 오유로 된다. 또한 상수형지적자가 아닌 지적자에 대하여 상수형지적자를 값주기하는것도 오유로 된다.

char *Ptr5 = Ptr4; // 틀림; 상수 Ptr4를 변경하게 한다.

C++언어

C++선언을 쉽게 해석하려면

C++문법을 보면 선언을 리해하기가 어렵다. 간단히 리해하는 방법은 선언을 뒤로부터 읽는것이다. 즉 오른쪽에서부터 왼쪽으로 가면서 읽어야 한다. 례를 들어 다음의 선언을 분석하시오.

char *const Ptr3 ;

오른쪽으로부터 시작한다면 《Ptr3은 char에 대한 상수지적자이다.》라고 할수 있다. 예약어 const는 《상수》라고 하며 *는 지적자라고 한다. 또 다른 실례를 들어 보자.

char const *Ptr;

Ptr1도 상수 char에 대한 지적자이다. 이러한 방법은 아무리 복잡한 선언도 쉽게 갈라 볼수 있게 한다.

11.4 배렬과 지적자

C++언어에서 배렬의 이름은 상수지적자로 볼수 있다. 즉 배렬의 이름은 구체적인 기억위치와 결합되여 있으며 이러한 결합은 프로그람에서 명령문에 의해서는 변화시킬수 없다. 실지 배렬의 이름은 배렬의 첫 요소의 기억위치라고도 볼수 있다. 실례로 아래의 6개의 명령문들을 들수 있다.

int A[5];
int B[10];
int *Ptr1 = A; //Ptr1에는 A[0]의 주소가 있다.
int *Ptr2 = B; //Ptr2에는 B[0]의 주소가 있다.
int *Ptr3 = &B[0]; // Ptr3 에는 B[0]의 주소가 있다.
int *Ptr4 = &A[4]; // Ptr4에는 A[4]의 주소가 있다.

처음 두개의 명령은 5개, 10개의 int형배렬 A와 B를 창조한다. 마지막 4개의 정의는 int*형의 지적자객체를 창조한다. 앞에서 본것처럼 Ptr1을 변수 A로 초기화한것은 배렬 A에서 첫 요소의 지적자와 같다. 또한 Ptr2의 정의는 배렬 B의 첫 요소를 가리키기 위하여 초기화한다. Ptr3의 정의는 B의 첫 요소를 가리키기 위해 초기화한다. Ptr4의 정의는 배렬 A에서 마지막요소를 지적하기 위해 초기화한다.

대입 및 관계연산자는 형이 같은 지적자에 대하여 정의할수 있다. 그러므로 앞서 정의된것처럼 다음의 명령문에서 비교는 성립될수 있다.

if (A == B)

```
        cout <<"A and B : same Values"<< endl;
    else
        cout << "A and B : different Values"<< endl;
    if ( Ptr1==A)
        cout << "A and Ptr1: same Values"<< endl;
    else
        cout << "A and Ptr1: different Values"<< endl;
    if (Ptr2 != Ptr3)
        cout << "Ptr2 and Ptr3: different Values"<< endl;
    else
        cout << "Ptr2 and Ptr3: same Values"<< endl;
    if (Ptr1 < Ptr4)
        cout << "Ptr1 and Ptr4: Ptr1 is first"<< endl;
    else
        cout << "Ptr1 and Ptr4: Ptr1 is not first"<< endl;
```

출력결과는 다음과 같다.

```
A and B: different Values
A and Ptr1: same Values
Ptr2 and Ptr3: same Values
Ptr1 and Ptr4: Ptr1 is first
```

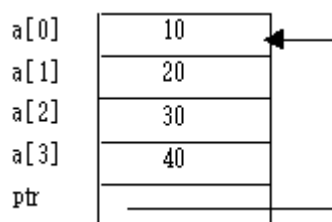
관계연산자 < ,<= ,> , >=가 같은 배렬이나 배렬다음에 기억기에 생기는 객체를 의미하는 지적자들에 대하여서만 정의된다는것을 참고하시오.

증가감소연산자는 지적자객체에 대하여 정의될수 있다. 지적자객체에 대하여 증가연산자 ++를 리용하는 경우는 이전에 지적하였던 객체다음의 시작위치를 가리키는 새로운 주소를 가지는것이다. 이와 같이 증가연산자는 반복자객체에 대한 조작과 류사한 방법으로 지적자객체에 대한 조작을 진행한다.

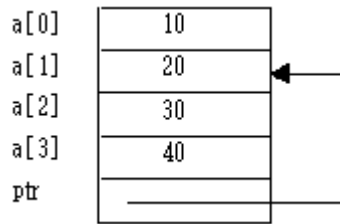
A와 Ptr에 대한 정의를 보시오.

```
int A[4] = {10, 20, 30, 40};
int *Ptr = A;
```

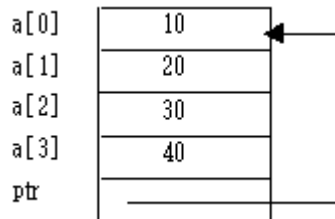
다음의 그림은 이 정의와 관련된 기억기에서 개별적인 값들의 모형을 보여 준다.

a[0]	10
a[1]	20
a[2]	30
a[3]	40
ptr	

이제 다음의 명령을 실행하면 cout흐름에 20이라는 값이 현시된다.

```
++Ptr;
cout << *Ptr << end1;
```

지적자증가연산자가 이전에 지적한 객체다음의 객체를 지적하기 위하여 연산수를 갱신하므로 이러한 결과가 얻어 진다. Ptr가 A[0]을 가리키므로 목록에서 다음요소를 지적하기 위하여 증가가 진행되며 다음요소는 A[1]이다. 아래의 기억기모형은 A와 Ptr사이의 관계를 보여 준다.

A[1]의 값이 20이므로 cout흐름에 *Ptr를 출력하면 20이 출력된다. 지적자객체에 대하여 감소연산자 --는 증가연산자와 반대로 동작한다. 만일 지적자객체가 감소하면 현재위치의 앞객체를 다시 지적하게 된다. 실례로 다음의 명령문은 현재의 Ptr를 감소시킨다.

```
--Ptr;
```

Ptr의 새로운 값을 A[0]의 위치로 할수 있다.

더하기, 덜기연산자들도 지적자객체에서 정의할수 있다. 실례로 식 Ptr + i는 Ptr를 지적하고 있는 객체로부터 i번째 객체에 대한 지적자이다.

주의

소홀히 할수 없는 표현오유

증가감소동작이 수행될 때 체계는 결과주소에 있는 객체가 지적자객체의 기본형과 같은가를 실행시에는 검열하지 않는다. 만일 형이 같지 않다면 프로그람은 정확히 수행되지 않는다. 실례로 다음의 프로그람을 보시오.

```
int A[5];
float x;
int *Ptr = &A[4]; //*Ptr는 A의 마지막요소를 지적한다.
++Ptr; //정의할수 없다. Ptr는 int위치를 지적할수 없다.
```

Ptr는 int형배렬에서 마지막요소를 의미한다. 대부분의 C++프로그람에서 float형객체 x는 기억기에서 A다음에 즉시 생긴다. 그러므로 Ptr의 순차적인 증가는 변수 x를 지적한다. 따라서 후에 Ptr를 사용하는 경우 예측할수 없는 결과가 생길수 있다.

이 식에서 i를 편위(offset)라고 한다. 이 편위주소에 있는 값은 왼쪽값 *(Ptr + i)를 리용하여 얻을수 있다. 또한 식 Ptr - i도 Ptr가 지적하는 객체의 i번째 객체에 대한 지적자이다. 이때 편위주소에 있는 값은 왼쪽값 *(Ptr - i)를 리용하여 얻을수 있다. 아래의 실례는 지적자참조해제연산자를 사용하여 배렬 A의 내용을 표시한다.

```
int A[4] = {10, 20, 30, 40};
```

```
int *Ptr = A;
for(int i = 0;i < 4; ++i){
    cout << *(Ptr + i) << endl;
}
```

지적자와 배렬사이의 일치성을 맞추기 위하여 C++에서는 식 *(Ptr+i)와 Ptr[i]가 같다고 본다. 따라서 배렬 A의 내용은 Ptr를 리용하여 다음과 같이 표시할수 있다.

```
for (int i = 0; i<4; i++){
    cont << Ptr[i] <<end1;
}
```

11.5 문자렬처리

iostream서고에는 오른쪽연산수가 char*객체인 출력과 입력연산자가 정의되여 있다. 프로그람작성자들이 문자렬에서 string클라스를 많이 사용하기때문에 다중정의의 의의가 적어 진다. 그러나 많은 서고들이 char형배렬이나 지적자표시를 사용하므로 이러한 입력과 출력연산자의 기능에 대한 론의는 여전히 중요하다.

출력흐름에 대한 char형지적자삽입결과는 첫 요소가 char*형객체가 지적하는 주소에 있는 char형배렬의 첫 요소를 표시하는것과 같다. 출력연산자의 사용실례를 아래에 준다.

```
char text[9]= "Sandrine";
for (char *Ptr = Text ; *Ptr ; =  ; ++Ptr){
    cout << Ptr << end1;
}
```

출력결과는 다음과 같다.

```
Sandrine
andrine
ndrine
drine
rine
ine
ne
e
```

for명령문이 반복, 수행되면서 Ptr가 지적하는 현재위치에서부터 시작하여 문자렬이 표시된다. 일단 Ptr가 빈 문자를 포함한 주소를 지적하면 검사식은 false로 되며 for명령문은 끝난다.

추출연산의 오른쪽연산수가 char*형객체이면 char형배렬에 대한 추출목적에 맞게 동작한다. 기정적으로 공백문자렬은 뛰여 넘게 되며 공백이 아닌 다음문자를 얻는다. 빈 문자를 포함한 완성된 문자렬로부터 얻은 문자들은 char*객체가 지적하는 기억기위치에서부터 배치된다. 다음의 정의는 이러한 실례이다.

```
char Word[4];
```

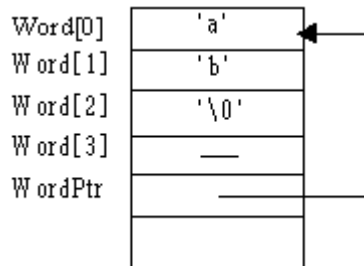
```
char *WordPtr = Word;
```

만일 표준입력이

```
abxyz
```

라면 추출명령

```
cin >> WordPtr;
```

의 결과는 아래의 기억기모형과 같다(가로는 그 객체가 초기화되지 않았다는것을 의미한다).

주의

필요한 기억구역이 있는가를 확인하시오.

char형지적자를 사용하여 추출을 진행하는 경우 지적자는 추출문자렬을 충분히 보관할수 있는 길이를 가진 char형배렬을 지적하여야 한다. 지적자는 자동적으로 쓸수 있는 기억구역을 충분히 확보한다. 만일 그렇지 않다면 프로그람에서 추출동작을 수행할수 없다. 아마 char형배렬다음에 기억된 자료값이 지워지게 될것이다. 운수가 좋으면 프로그람은 추출명령문근방에서 오유가 생기게 될것이며 그 경우 리유는 명백하다. 그러나 때때로 오유가 이 명령문과 멀리 떨어 진 위치에서 생기기때문에 배렬은 지내 많이 쓰게 된다.이때 오유를 찾는다는것은 어려운 일이다.

함수에 대한 문자렬의 경로에서 지적자와 배렬표의 교환은 가장 뚜렷하다. 다음의 함수는 cstring서고의 strlen()함수를 실현한것이다.

```
// const 배렬을 통과하는 s를 가진 strlen()
int strlen(const char s[]) {
    int i;
    for(i = 0; s[i] != '\0'; ++i) {
        continue;
    }
    return i;
}
```

함수는 빈 문자로 끝나는 하나의 파라메터로서 문자렬을 요구하며 문자렬의 길이 (빈 문자만은 제외한다.) 를 돌려 준다. 문자렬의 길이는 빈 문자가 나타날 때까지 문자렬을 주사하여 결정한다. 빈 문자에 대한 첨수는 문자렬의 길이이다(문자렬은 배렬요소로서 s[0]부터 s[i-1]까지이다).

strlen()함수는 또한 첫 문자에 대한 지적자로 넘겨 진 문자렬을 가지고 다음과 같은 방법으로 실현할수 있다.

```
// const 문자에 대한 지적자를 통과한 s를 가진 strlen()함수
int strlen(const char *s) {
```

```
        int length;
        for (length = 0; *s != '\0'; ++s) {
            ++length;
        }
        return length;
    }
```

값파라메터 s의 **const**선언은 s가 지적하는 위치의 내용이 상수임을 의미한다. 즉 *s는 값주기대상이 될수 없다.문자렬의 길이는 비지 않은 매 문자들에 대하여 **int**형객체 length를 증가시켜 결정한다. 문자렬의 다음문자를 지적하도록 하기 위하여 s를 갱신한다. strlen()함수를 다르게 실현할수 있다. 이 방법은 다음문자에 대한 지적자를 증가시키는 방법으로 문자비교를 결합하였다.

```
    // strlen()은 파라메터로서 상수문자에 대한 지적자를 통과하는 s를 가진다.
    // 지적자의 증가는 시험식의 결과로서 진행된다.
    int strlen(const char *s) {
        int length;
        for (length = 0; *s ++ i = '\0' ; ++length) {
            continue ;
        }
        return length;
    }
```

이 시험식은 다음과 같이 분석할수 있다. s가 지적하는 **char**형객체의 값이 빈 문자인가를 알아 보고 다른 한편 지적자 s를 증가시킨다. 이러한 평가와 지적자증가의 결합은 문자렬처리코드에서 가장 일반적이다. 그러나 그것이 리해에 도움이 되겠는지 알수 없다.

다음 **int**형함수 strcmp()를 볼수 있다. 이 함수는 2개의 파라메터를 가지고 있으며 돌림값은 두 문자렬이 같은가, 같지 않은가 하는 값이다.

```
    int strcmp(const char *s, const char *t)
        // *s가 빈 문자일 때까지 첫번째 다른 문자를 찾는다.
        while (c*s==*c) &&(*s|='/0')) {
            ++s;
            ++t;
    } // 그 차이를 구하여 되돌리기
        return * s - * t;
    }
```

strcmp()함수의 문자렬파라메터는 s와 t이다. 함수 strcmp()는 문자렬이 같다면 0을 되돌리고 s가 t이전에 놓인 문자라면 부수를, s가 t후에 놓인 문자라면 정수를 되돌린다. strcmp()에서 **while**순환은 2개 조건이 참일 때까지 반복된다. s와 t가 지적하는 현재문자의 값이 같고 s가 지적하는 현재문자의 값이 빈 문자가 아닐 때까지 반복된다. 순환이 끝나면 함수의 돌림값이 결정된다. 만일 문자들이 같다면 돌림값의 결과(*s-*t)는 0으로 된다. 문자들이 서로 다르다면 두 문자렬에서 제일 처음으로 차이나는 문자를 가리키게 된다. 이 경우에 s가 t전에 있다면 식 (*s- *t)는 부수값을 가지며 s가 t다음에 있다면 정수값을 가진다.

11.6 프로그람지령행파라메터

많은 조작체계(UNIX, Windows)들에는 지령해석기가 있다. 이 해석기들은 사용자로부터 지령을 받아 조작체계가 그것을 실행하게 한다. 실례로 다음의 명령은 현재등록부를 등록부 code로 바꾸는 cd 지령이다.

```
cd code
```

이 명령에서 문자렬 code는 지령 cd의 파라메터이다. 지령은 두개의 문자렬 cd와 code로 이루어 졌다.

또 다른 실례가 있다. 다음의 구조는 지령행파라메터로서 문자렬 123과 ab를 가진 프로그람 cmd를 실행한다.

```
cmd 123 ab
```

C++에서는 지령행파라메터를 리용하여 프로그람을 작성할수 있다. 지령행파라메터는 main()함수를 통하여 프로그람에 넘어 간다. 이를 위하여 C++는 프로그람이 지령행파라메터들을 문자렬로서 고찰하도록 한다. 첫번째 문자렬은 프로그람이름이고 나머지문자렬들은 프로그람의 파라메터이다. 두번째 실례와 같이 순서대로 있는 3개의 문자렬 cmd, 123, ab를 들수 있다.

C++는 지령행을 구성하는 문자렬을 호출하기 위하여 두개의 파라메터를 규정하는 main()함수를 위한 파라메터목록선언을 한다.

```
int main(int argc, char *argv[]);
```

첫번째 파라메터는 int형파라메터이다. 이 파라메터는 습관적으로 argc라고 한다. 프로그람이 실행될 때 파라메터 argc는 자동적으로 지령행에 놓인 문자렬의 개수로 초기화된다. 이 수는 주어 진 개수만한 프로그람의 이름을 가지며 argc는 프로그람이름에 대한 지령행파라메터+1개의 수이다. 우의 cmd실례에서 argc의 값은 3으로 된다.

두번째 파라메터는 char형지적자배렬이다. 약속한것처럼 이 배렬은 argv이다. 프로그람이 실행될 때 지적자 argv[0]은 자동적으로 프로그람이름을 의미하는 문자렬을 가리키기 위하여 초기화되며 지적자 argv[1]부터 argv[argc-1]은 프로그람의 개별적지령행파라메터를 표시하는 문자렬을 표시하기 위하여 초기화된다. 이 초기화는 그림 11-2의 cmd실례에 의하여 설명된다.

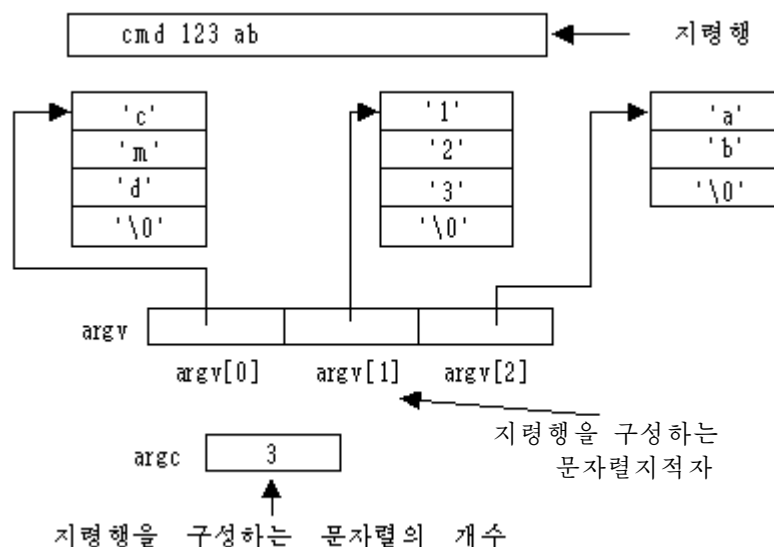



그림 11-2. 지령행파라메터와 main()함수파라메터 argc , argv와의 대응

argc와 argv가 main()함수의 파라메터라는것이 중요하다. 그것들은 전역객체가 아니다. 만일 실행시에 다른 함수가 argc와 argv값을 요구한다면 argc와 argv는 파라메터로서 그 함수들을 통과하여야 한다.

프로그람 11-2는 표준출력흐름에 지령행파라메터를 표시하는 간단한 프로그람이다. 프로그람을 통과하는 지령행파라메터들은 매 지령행파라메터들을 위하여 한번 반복하는 for순환명령에 의하여 련속 현시된다. 개별적인 파라메터는 삽입연산자를 리용하여 표시한다. 11.5에서 취급한것처럼 삽입연산자는 char*지적자로 다중정의되는데 지적된 문자렬이 현시된다.

```
// 프로그람 11-2: 조작체계지령 echo를 모방하는 실례
// 표준출력흐름에 공백으로 분리된 지령행파라메터들을 현시한다.
#include <iostream>
#include <string>
using namespace std;
int main(int argc, char *argv[]) {
    for (int i = 1; i <argc; i ++) {
        cout << argv[i] << ""; // argv[i]를 현시
    }
    cout << endl;
    return 0;
}
```

프로그람 11-2. 조작체계지령 echo를 모방하는 실례

프로그람 11-3도 역시 지령행파라메터를 리용한것이다. 이 프로그람은 파라메터를 파일이름으로 요구한다. 매 파라메터는 입력파일을 열기 위해 사용된다. 파일흐름의 내용은 그때 표준출력흐름으로 표시된다. 프로그람은 파일흐름서고 fstream을 사용한다.

```
// 프로그람 11-3: 조작체계지령형을 모방하는 실례
// 지령행파라메터는 파일이름이다. 관련파일의 내용을 표준출력흐름 cout에 현시한다.
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main(int argc, char *argv[]) {
    ifstream fin;
    for (int i = 1; i < argc; ++i) {
        fin.open(argv[i]); // i 번째 지령행파라메터에 련결된 파일흐름열기
        if (fin) {
        char c; // argv[i]가 파일이름이므로 파일안에 있는 문자들을 입력하여 표시한다
        while (fin.get( c )) {
```

```
            cout << c;
        }
        else {
        // argv[i]가 파일이름이므로 오유통보를 현시하고 오유통보를 내보내고 완료한다.
            cerr << argv[0] << ":"<<argv[i]
              <<"not a valid file name"<< endl;
            return 1;
        }
        fin.close();
    }
    return 0;
}
```

프로그람 11-3. 조작체계지령형을 모방하는 실례

프로그람 11-2와 같이 프로그람 11-3에도 파라메터의 처리를 조종하는 for순환명령이 있다. 만일 ifstream객체 fin이 argv[i]라는 이름을 가진 파일을 성과적으로 열면 문자들을 하나하나 추출한다. 매 문자들이 추출되므로 그것은 표준출력흐름으로 표시한다. 파일을 열수 없으면 오유통보가 나타난다. 오유통보는 잘못된 파일이름을 파라메터로 하여 실행된 프로그람의 이름을 포함한다.

문 제

13. 다음과 같은 정의가 있다.

```
int GradeList[100];
int *Ptr;
```

Ptr가 GradeList의 첫 요소를 가리키게 하는 값주기명령을 작성하시오. 또한 GradeList의 마지막요소를 Ptr가 가리키게 하는 값주기명령을 작성하시오.

14. 다음과 같은 C++프로그람이 있다.

```
char *s;
char p[20] = "Happy Holidays";
s = &p[3];
cout << s[4] <<end1;
```

이때 출력하는 값은 무엇인가?

15. 다음과 같은 정의가 있다.

```
int Weight[100];
int *Ptr;
```

Weight배렬의 요소에 접근하기 위하여 Ptr를 리용하고 배렬 Weights의 매 요소에 0을 대입하는 순환명령문을 쓰시오.

16. 다음과 같은 정의가 있다.

```
int A[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, };
```

AR의 첫 요소가 초기화되여 A의 마지막요소를 지적하도록 초기화하는 AR옹근수지적자배렬을 정의하는 명령문과 AR의 두번째 요소가 A의 마지막 다음요소를 지적하게 하는 명령문을 쓰시오.

17. src라는 주소로부터 배치된 n개의 옹근수배렬을 주소 dst라는 새로운 위치에 복사하는 WordCopy함수를 만드시오. WordCopy함수의 형태는 다음과 같다.

```
void WordCopy(int *dst, int *src, int n);
```

18. WordCopy의 원형이 다음과 같이 변화되였다.

```
void WordCopy(int *dst, const int *src, int n);
```

복사함수를 계속 쓸수 있는가? 그 리유는 무엇인가 ?

19. WordCopy의 원형이 다음과 같이 변화되였다.

```
void WordCopy(const int *dst, const int *src, int n);
```

복사함수를 계속 쓸수 있는가? 그 리유는 무엇인가?

20. WordCopy의 원형이 다음과 같이 변화되였다.

```
void WordCopy (int *dst, int *const src , int n);
```

복사함수를 계속 쓸수 있는가? 그 리유는 무엇인가?

21. number라고 하는 프로그람을 작성하시오. number프로그람의 지령행은 다음과 같다.

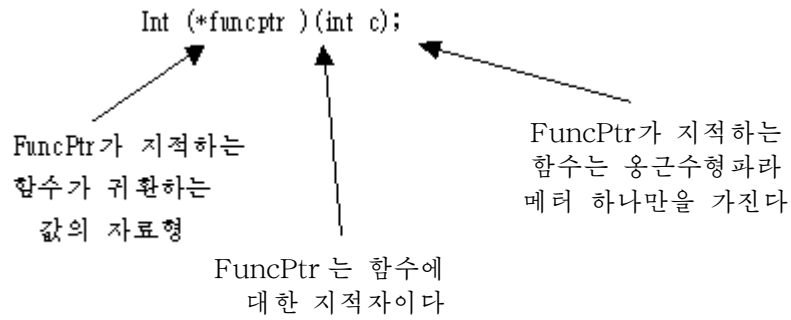
```
number  filename1 filename2
```

number프로그람은 filename1을 filename2에 복사하며 매행에 해당한 번호를 출력한다. number프로그람의 출력내용은 다음과 같다.

```
0001: 이것은 첫행이다.
0002: 이것은 두번째 행이다.
...
```

11.7 함수에 대한 지적자

C++에서는 객체에 대한 지적자와 함수에 대한 지적자도 리용할수 있다. 일반적으로 함수에 대한 지적자는 다른 함수에 대한 파라메터로서 쓰인다. 사용한 함수가 여러가지 동작을 수행할수 있게 한다. 이러한 능력은 객체지향언어가 아닌데서는 중요하지만 C++와 같이 객체지향언어에서는 얼마 리용되지 않는다(14장을 보시오).

함수에 대한 지적자형도 기본형이다. 그러므로 함수에 대한 지적자인 객체에 값주기할수 있다. 또한 함수의 귀환값형도 함수에 대한 지적자로 될수 있으며 함수에 대한 지적자배렬도 정의될수 있다. 함수에 대한 지적자로 되는 객체의 정의에서는 함수의 귀환값형과 함수의 파라메터형을 포함하여야 한다(읽을수 있게 파라메터의 이름도 제공될수 있다). 아래에 한개의 int값파라메터를 요구하는 int함수에 대한 지적자인 FuncPtr객체를 정의한다.

배렬이름이 그의 요소에 대한 지적자인것만큼 함수이름 역시 자기의 동작을 구성하는 코드에 대한 지적자이다. 그러므로 다음의 값주기명령은 int형의 함수 toupper()에 대한 지적자인 FuncfPtr를 결과로 준다.

FuncPtr = toupper;

프로그람 11-4는 함수에 대한 지적자의 리용실례를 보여 준다. 프로그람은 먼저 입력재촉문을 쓰고 파일의 이름을 추출한다. 다음 파일의 내용을 대소문자로 표시하는가 하는 사용자의 요구를 얻는다. 그러면 Display()함수가 동작한다. Display()함수는 두개의 파라메터를 요구하는데 첫 파라메터 fin은 입력파일흐름이고 두번째 파라메터 ToFunc는 입력흐름에서 문자를 처리하는 함수에 대한 지적자이다.

```
// 프로그람 11-4: 사용자의 요구에 따라 대문자 혹은 소문자로 파일을 현시한다.
#include <fstream>
#include <ctype.h>
#include <string>
using namespace std;
// Display(): ToFunc를 리용하여 본문흐름 현시
void Display(ifstream &fin, int (*ToFunc)(int c)) {
    // 한번에 한 문자씩 추출하여 현시한다.
    char CurrentChar;
    while(fin.get(CurrentChar)) { // ToFunc를 리용하여 현재문자를 변경한다.
        CurrentChar = (*ToFunc)(CurrentChar);
        cout << CurrentChar;
    }
    return;
}
int main() { // main():파일흐름표를 관리한다.
    const int MaxFileNameSize = 256;
    // 파일이름을 요구하고 얻는다.
    cout << "Enter name of file:"<< flush;
    char FileName [MaxFileNameSize];
    cin >> FileName;
    ifstream fin(FileName); // 파일이름이 옳은가 확인한다.
```

```
    if (fin) { // 파일이름이 옳으면 동작을 결정한다.
        cout << "Display file in uppercase or "
          <<"lowercase (u,l):"<< flush;
        char reply;
        cin >> reply;
        if (reply == 'l') // 요구에 따라 파일을 현시
            Display(fin, tolower);
        else if (reply == 'u')
            Display (fin, toupper);
    else {
        cerr << "Bad request "<< endl;
        return 1;
        }
    }
    else{ // 파일이름이 틀렸을 때 처리
        cerr << "Invalid file name: "<< FileName << endl;
        return 1;
    }
    return 0;
}
```

프로그람 11-4. 파라메터로서 함수지적자를 리용하는 실례

함수 toupper()와 tolower()는 main()함수안에서 display()의 호출시 본문처리함수파라메터로 리용된다. 함수 toupper()와 tolower()는 ctype서고에 정의되여 있으며 그것들이 문자처리를 위하여 설계되였다고 하더라도 그것들의 파라메터와 귀환값형은 **int**로 정의된다. ctype서고는 C의 기초표준서고이므로 선언되는 파일은 ctype이다. 파라메터선언과 류사한 문법이 Display()함수에서 ToFunc를 호출하는데 리용된다.

```
CurrentChar = (*ToFunc)(CurrentChar );
```

호출에서 파라메터 CurrentChar는 입력지령으로부터 추출된 현재의 문자들을 읽는다. CurrentChar는 함수호출결과를 얻는것으로써 변경된다. 더 간단한 호출문법이 있다. 다음의 호출에서와 같이 괄호와 참조해제연산자는 생략할수 있다.

```
CurrentChar = ToFunc(CurrentChar);
```

콤퓨터의 력사

Altair 8800

소편집적기술이 발전함에 따라 인텔회사는 강력한 소편들을 대대적으로 생산하였다. 인텔은 1974년에 8080극소형처리소자를 생산하였다.8080의 능력은 당시의 일부 소형콤퓨터들과 대등한 위치에 있었다. 많은 전자공학애호가들과 발명가들은 이 기회를 놓치지 않았다. 에드워드 로버트는 뉴 멕시코의 앨부쿼쿠에 있는

전자제품을 판매하는 작은 회사를 차려 놓았다. 로버트는 8080을 도입하여 가정용콤퓨터를 만들고 판매할수 있는 기회를 얻게 되였다. 이 콤퓨터는 잡지 《Popular Electronics》의 1975년 1월호 표지에 소개되였다. 이 기계를 Altair 8800이라고 하였다. Altair라는 이름은 대중 TV계렬 Star Trek의 최근 일화가 그 이름을 대기업화의 목표로 리용한데로부터 쓰이게 되였다. 또한 과학환상영화 《금지된 행성 》에서 행성의 이름과 같다. Popular Electronics의 표지에 나타난 Altair 8800은 개인용콤퓨터혁명에 불을 지펴 주었다.이 기사는 콤퓨터로 작업하고 쏘프트웨어를 작성하는 콤퓨터애호가들을 격동시켰다.그들중 일부는 오늘도 콤퓨터산업을 형성하고 있는 회사들을 창설하는데 특출한 기여를 하였다(실례로 마이크로쏘프트의 파울 알렌과 윌리암 게이츠).

그림 11-3. Altiar 8800 을 소개한 Popular Electronic 1975 년 1 월호

문 제

22. 함수의 원형이 다음과 같다.

```
void (*signal (int sig, void (*func) ( int )))( int );
```

이 함수는 어떤 기능을 수행하는가를 설명하시오. 오른쪽으로부터 왼쪽으로 가면서 함수정의지적자를 읽어 보시오.

23. 다음의 프로그람의 출력결과는 무엇인가?

```
#Include<iostream>
void Func1(int Value) {
    cout << "Func1 says"<< Value << end1;
    return ;
}
void Func2(int Value) {
    cout << "Func2 says"<< Value << end1;
    return;
}
int main() {
```

```
void (*FuncPtr)(int);
    funPtr = &Func1;
    (*FuncPtr)(17);
    FuncPtr =&Func2;
    (*FuncPtr)(42);
    return 0;
}
```

24. Apply이라는 함수를 작성하시오. Apply함수는 옹근수배렬과 그 배렬의 요소수, 옹근수값을 돌려주는 함수에 대한 지적자를 파라메터로 가진다. 함수 Apply는 배렬의 매 요소를 통과시켜 함수를 적용한다.

11.8 동적객체

앞에서 서술한것처럼 지적자와 객체를 관리하기 위하여 참조해제연산자와 주소연산자를 리용할수 있다. 지적자가 배렬관리에 리용되며 또한 C++프로그람인 지령행파라메터를 호출할수 있게 하는 기구라는 것을 알수 있다. 지적자표시와 지령행토막에 대한 호출이 다 중요하지만 이것은 지적자가 C++언어의 한 부분으로 되여야 한다는 기본리유로는 되지 못한다. 지적자의 기본역할은 동적객체를 창조하는데 있는데 프로그람이 실행될 때 기억기의 변화를 요구하는 객체에서 특히 더하다. 구체적으로 표준본보기서고(STL)는 동적객체를 리용하여 그러한 기능을 지원한다.

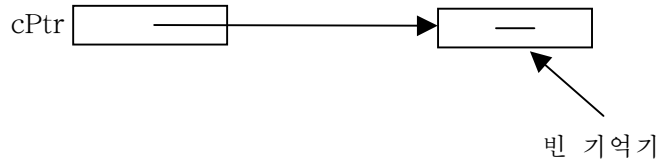
cout, cin 그리고 일부 전역상수를 비롯한 전역흐름객체들을 제외하고 프로그람에서 정의되는 객체들은 다 국부객체이다. 국부객체들은 정의가 될 때만 존재하며 정의된 블로크유효범위를 벗어날 때는 자동적으로 없어 진다. 전역객체는 프로그람이 시작될 때 유효범위의 시작부터 존재하기 시작하며 프로그람이 완료될 때(유효범위의 끝) 자동적으로 없어 진다.

동적객체는 프로그람에 의하여 요구되므로 존재에서 국부, 전역객체와는 다르다. 동적객체는 프로그람이 지정된 크기의 기억기를 해방할것을 요구할 때까지 존재한다. 따라서 동적객체의 존재시간은 정의된 유효범위에서 독립적이다.

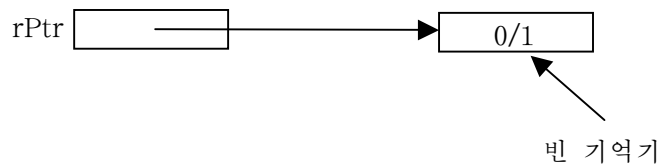
동적객체에 의하여 리용되는 기억기는 빈 기억기에 할당된다고 할수 있다.빈 기억기는 조작체계에 의하여 필요할 때마다 프로그람에 할당도 하고 해방도 할수 있는 기억기이다. C++에서 빈 기억기의 사용방법은 단항연산자 new를 통하여 제공된다. 동적객체가 더이상 필요없을 때 기억기는 단항연산자 delete를 리용하여 빈 기억기로 돌려 진다.

new연산자는 3개의 형태를 가진다. 가장 간단한 형태는 new연산자에 대한 오른쪽연산수의 형이름을 요구하는 하나의 객체에 대한 기억기할당요구이다. 할당되지 않은 빈 기억기가 충분하다면 조작은 주어 진 기억기위치에 대한 지적자를 돌려 준다. 아래에 이러한 실례를 주었다.

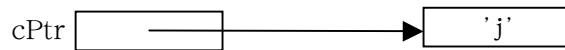
```
char *cPtr ;
cPtr = new char ; // cPtr는 초기화되지 않은 char객체를 지적한다.
```

우의 명령에서 첫번째 행은 char*형객체 cPtr를 정의하였다. 만일 빈 기억기를 쓸수 있다면 값주기명령문은 빈 기억기의 이전 부분이던 char 객체기억기위치에 cPtr를 설정한다. char객체를 위한 기억기는 기본형객체에 대한 구축자가 없기때문에 초기화되지 않는다.

다음의 실례와 같이 빈 기억기로부터 요구되는 객체형이 클라스형이라면 새로 할당된 기억기를 초기화하기 위하여 클라스가상구축자가 자동적으로 호출된다.

```
Rational *rPtr;
rPtr = new Rational ; // rPtr는 0과 1로 표현되는 Rational객체를 가리킨다.
```

우의 값주기명령문실행에 의하여 가상값 0과 1로 표시되는 동적Rational객체를 가리키는 Rational 지적자 rPtr가 생긴다.

지적자가 정확하다면 new요구에 의하여 돌려 지는 위치로 설정하며 그 위치에서 객체는 주어 진 다른 객체처럼 리용할수 있다. 구체적으로 말한다면 평가도 할수 있고 관리도 할수 있다. 다음의 명령에서는 cPtr가 지적하는 동적객체의 값이 다음입력문자값으로 설정된다.

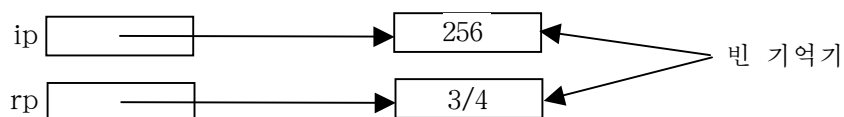
```
cin >> *Ptr // cPtr가 지적하는 char객체에 입력된 문자를 넣어 준다.
```

다음으로 입력된 문자가 j라고 가정하자. 그러면 문자 j가 입력된후 기억기는 다음과 같이 된다.

결과 다음의 삽입명령은 j를 cout로 현시한다.

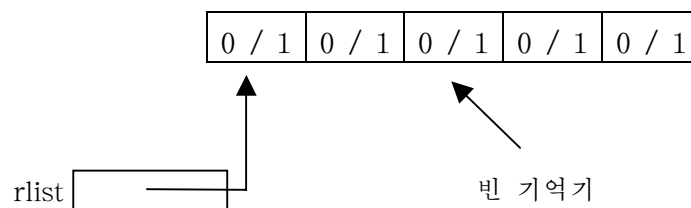
```
cout << *cPtr ; // cPtr가 지적하는 char형객체를 표시한다.
```

new연산자의 두번째 형태는 객체에 대하여 하나의 동적객체를 요구할뿐아니라 초기화도 할수 있다. 초기화값들은 괄호안에서 반점으로 분리하여 지정할수 있다. 다음의 명령에서 지적자 ip와 rp는 지적된 값으로 초기화를 진행한다.

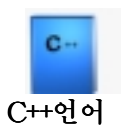
```
int *ip;              //ip는 256개의 값을 표시하는 동적객체를 지적한다.
ip = new int (256);
Rational *rp;         //rp는 4와 3으로 된 동적객체를 지적한다.
rp = new Rational(3, 4);
```

만일 충분한 빈 기억기가 존재한다면 ip는 256개의 값을 표시하는 동적int형객체를 지적하며 rp는 값 3과 4를 표시하는 Rational형객체를 지적한다.

new연산자의 세번째 형태는 한개 연산에서 여러개의 동적객체를 요구하게 하는것이다. 이 형태는 주어 진 객체의 수를 표시하는 첨수가운데서 지적하는데 따라 new연산자가 형이름인 오른쪽연산수를 요구한다. 할당되지 않은 빈 기억기가 충분히 남아 있다면 연산은 요구되는 객체의 수를 얻는데 충분한 동적기억블로크에 대한 지적자를 돌려 준다. 만일 요구되는 동적객체의 형이 클라스형이라면 요구되는 매 객체를 초기화하기 위하여 가상클라스구축자가 자동적으로 호출된다. 이런 형태를 다음과 같이 쓸수 있다.

```
Rational *rlist;
rlist = new Rational[5];
```

이 값주기명령에서 new연산이 정확히 진행된다면 rlist는 5개의 Rational객체를 얻는데 충분한 기억기블로크에서 첫 객체를 지적하게 된다.

기억기가 서로 련결되였기때문에 지적자 rlist는 5개의 요소로 된 Rational배렬로 볼수 있다. 매 동적객체는 Rational가상구축자로 초기화되는데 즉 5개의 Rational객체들이 모두 0파 1이라는 유리수값으로 표시된다.

지금까지 설명한 rlist실례는 여러개의 요구하는 동적객체에서 한개 상수식을 리용하였는데 상수식은 요구가 없다. 실례로 다음의 코드토막에서는 cin함수로부터 값을 추출하여 요구하는 빈 기억기의 크기를 결정한다.

C++언어

new연산과 례외처리

지금까지 new조작에 대한 론의에서는 빈 기억기의 요구가 만족되지 않을 경우 어떤 현상이 일어나겠는가를 고찰하지 않았다. 만일 요구가 만족되지 않는 경우 C++는 표준례외처리를 진행한다. 그러나 현재의 많은 프로그람들에서는 요구가 만족될수 없다면 조작결과가 0 즉 빈 주소를 만든다.

례외는 실행시 생기는 프로그람오유이다. 례외처리기코드토막이 정의되여 있다면 조종흐름은 례외가 생길 때 그 처리기로 넘어 간다. 처리기가 없다면 프로그람은 례외가 생길 때 완료된다. 프로그람작성자들은 또한 산수적인 오유(실례로 령으로의 나누기 등)에 대한 례외를 검출하고 조종하는 함수를 흔히 작성한다.

C++례외처리기구는 만족하지 않는 여러가지 빈 기억기요구에 대하여 각이한 례외처리기를 작성할수 있다. 응용프로그람에 따라 이러한 유연성은 쓸모 있다. C++는 또한 new조작결과가 0이 되는 앞의 동작을 얻기 위한 방법을 제공한다. 특히 new서고를 제공하는것이다. 이 서고는 new연산자의 례외처리함수에 대한 지적자를 파라메터로 요구하는 void형함수 set_new_handler()를 정의한다. set_new_handler(0)에서와 같이 실제파라메터로 0을 사용한다면 그때 불만족한 new연산에 대하여 0을 돌려 주는 앞의 동작에 적용된다.

자체의 례외처리함수를 실현하는 방법은 이 책에서 주지 않는다. 간단한 소개는 부록 4에 있다.

```
cout <<"Size of list"<<flush;
int ListSize;
```

```
cin >> ListSize ;
int *Values = new int [ListSize];
```

한개의 객체를 리용하는 빈 기억기로부터 얻어 진 기억기는 delete연산의 오른쪽연산수와 같이 그 기억기에 대한 지적자를 제공하는것으로써 빈 기억기에 돌려 줄수 있다. 다음의 프로그람은 cPtr가 지적하는 기억기를 돌려 주는 동작을 수행한다.

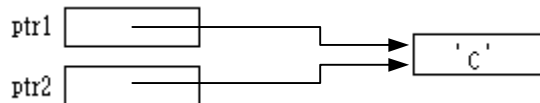
```
delete cPtr ;
```

delete연산의 한 측면은 cPtr값이 정의되여 있지 않다는것이다.

cPtr —

빈 기억기로 돌려 지는 기억기의 위치에 대한 호출이 정확치 않으므로 프로그람작성자들은 자기가 작성한 프로그람에서 같은 객체에 여러개의 지적자가 있는가를 주의깊게 살펴야 한다. 실례로 다음의 코드를 분석하시오.

```
char *Ptr1 = new char ('c');
char *Ptr2 = Ptr1;
delete  Ptr1;
cout << *Ptr2 ; //정의되지 않은 결과 Ptr2는 되돌려 진 빈 기억기를 지적한다.
```

이 코드는 2개의 char*형지적자 Ptr1과 Ptr2의 정의로 시작된다. 지적자 Ptr1은 값이 c인 동적char형객체의 위치에 대한 new연산을 초기화한다.지적자 Ptr2는 같은 객체를 지적하여 초기화된다.

두개의 정의다음에 있는 delete명령은 빈 기억기에 'c'를 표시하는 기억기를 돌려 준다. 결과 Ptr1의 값은 정의되지 않는다. 또 다른 경우 Ptr2는 예속지적자(dangling pointer)로 되는데 Ptr2는 현재 필요 없는 기억기의 위치를 지적한다.

빈 기억기에로 돌려 지는 기억기호출은 정의되지 않는 결과를 만든다. 빈 기억기요구에서 객체의 수를 의미하는 new형을 리용하여 얻은 동적객체는 예약어 delete와 돌려 지는 기억기에 대한 지적자사이에 한쌍의 팔호를 포함한 delte형을 리용하는 한개 배렬로서 돌려 져야 한다. 삭제된후 지적자값은 정의되지 않는다.

다음의 명령에서 delete연산은 rlist가 지적하는 5개의 Rational동적객체의 기억기를 빈 기억기에 돌려 준다.

```
delete [] rlist ;
```

만일 빈 기억기에 돌려 지는 기억기가 클라스형객체를 표시하고 해체자클라스가 있다면 해체자는 자동적으로 매 객체들에 대하여 호출된다. 해체자는 소멸되는 클라스객체에 대하여 필요한 처리를 진행하는 성원함수이다. 앞의 장에서 정의한 클라스(Rational, Maze)들에는 이러한 작용들이 요구되지 않으며

해체자는 그 클라스안에 정의되여 있지 않았다.

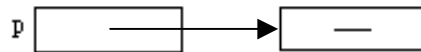
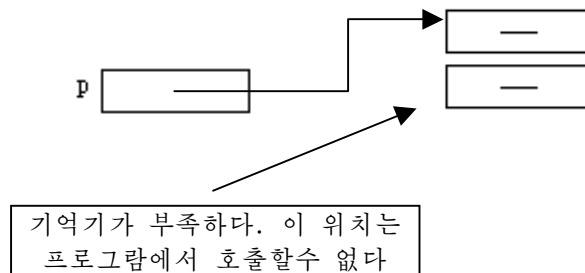
주의

오유가 생기면

예속지적자를 리용한 효과는 즉시에 나타나지 않는다. 그러므로 동적객체를 리용하는 프로그람이 잘못된것이 명백하다면 자료구조에 대한 매 삭제효과를 다시 생각해 보아야 한다. 다시 말하여 무엇이 생기는가를 그림으로 그리는것은 프로그람오유를 찾는데서 매우 효과적이라고 할수 있다.

해체자는 대체로 동적기억기에 대한 지적자를 가진 자료성원클라스객체에만 필요하다. 왜냐하면 자기의 자료성원에 대한 delete연산을 실시하는것이 해체자이기때문이다. 동적기억과 보관한 변수의 위치를 지적자 p에 값주기하는 코드가 아래에 있다.

```
int *p;
P=new int ; //p는 동적기억기를 지적한다.
P=new int ;//p는 현재 다른 기억기를 지적한다.
```

처음에 값주기명령문이 수행되면 빈 기억기가 얻어 지고 p는 빈 기억기를 지적한다.

두번째 값주기명령이 수행되면 보충적인 빈 기억기가 얻어 지고 p는 현재 새로 생긴 빈 기억기를 지적한다.

이 프로그람에서 이전에 얻어 진 기억기는 없어 진다. 처음에 진행된 new연산에 의하여 얻어 진 기억기의 위치값을 가지는 지적자객체가 없으므로 자료루실이 생긴다. 이 자료루실을 기억기루실(memory leak)이라고 한다. C++프로그람에서는 이러한 현상이 자주 나타난다. 기억기루실은 프로그람이 실행될 때 큰 영향을 주게 된다. 왜냐하면 빈 기억기가 충분하지 못한것으로 하여 다음에 진행되는 동적기억기요구를 원만히 보장할수 없게 하기때문이다. 그러므로 동적기억기요구를 처리할 때 기억기루실에 항상 주의를 돌려야 한다.

11.9 옹근수값의 목록을 표시하는 간단한 ADT

옹근수값목록을 표시하는 간단한 클라스인 IntList를 만들어 보자. 이 클라스는 동적기억기할당과 재할당을 할수 있게 한다. IntList클라스의 개발은 vector와 다른 용기클라스들이 어떻게 동작하는가를 알수 있게 해준다. 본보기와 다형성을 론의할 때 14장에서 용기클라스의 개발에 대하여 다시 볼수 있다.

IntList클라스의 첫 대면부는 배렬대면부와 류사하다. 목록의 개별적요소를 호출하는 왼쪽값과 오른

쪽값을 제공하는 첨수연산자를 다중정의하였다(배렬과 같이 IntList요소는 번호 0부터 시작된다). IntList클라스는 표준배렬의 결함들을 극복하였다. 구체적으로 IntList객체는 값주기의 원천이나 대상으로도 되며 참조와 평가를 둘다 할수 있고 함수에 대한 값을 돌려 줄수 있으며 초기에 지정한 초기값과 요소값을 고려하는가 가상구축자를 가진다. 그리고 목록에 표시된 요소의 수를 호출할수 있으며 첨수연산자를 다중정의한 IntList는 색인값이 적당한 범위내에 있는가를 확인하기도 한다. 리용하기 쉬운 IntList클라스를 만들기 위하여 IntList객체에 삽입연산자를 다중정의한다. IntList추출연산자와 반복자가 있다고 가정한다. 아래의 코드에서는 여러가지 목록을 정의하고 처리하기 위한 IntList클라스를 사용하였다.

```
IntList A ; //기정으로 목록의 내용이 0인 10개의 목록
cout << "A: "<<A <<end 1;
IntList B(5.1);//목록의 내용이 1인 5개의 목록
cout << "B:"<<B <<end1;
cout << "Number of values:";
int n;
IntList C(0.2): //목록의 내용이 2인 n개의 목록
cout << "C:"<< C <<end1;
B = A;
for (int i=0; i<A.size() ; ++i) { //A를 변경시킨다.
    A[i]=i ;
}
cout << "A: "<< A << end1;
cout << "B: "<< B << end1;
```

입력재촉문을 리용하여 프로그람에 값 3을 넣으면 입출력결과는 다음과 같다.

```
A:[0 0 0 0 0 0 0 0 0 0]
B:[1 1 1 1 1]
Number of values: 3
C:[2 2 2]
A:[0 1 2 3 4 5 6 7 8 9]
B:[0 0 0 0 0 0 0 0 0 0]
```

초기코드토막과 초기출력결과에서와 같이 기정구축자는 10개의 요소로 된 목록 A를 0이라는 값으로 만든다. 토막과 출력은 요소(5)의 수와 그 요소(1)의 값을 지정하여 목록 B를 작성할수 있다는것을 보여 준다. C의 정의와 현시는 실행이 끝날 때까지 구축된 목록의 크기값을 알수 없다는것을 보여 준다. 콤파일될 때만이 그 크기를 명백히 알수 있다.

두번째 A와 B의 현시값은 B에 A의 값을 대입하고 A의 값을 변경시킨후에 생긴다. A와 B는 변할수 있는 객체들인데 다른 객체에 영향을 주지 않으며 IntList객체에서는 요소의 수와 값을 변경시킬수 있다.

11.9.1 IntList의 명세부

목록 11-1에 있는 IntList의 클라스정의는 IntList의 소개에서 설명한 동작과 능력을 표시하는 실행구조를 보여 준다.

목록 11-1. intlsit.h에 정의된 IntList클라스

```
class IntList {
    public:
        // 기정구축자
        IntList (int n = 10, int val = 0 );
        // 구축자를 표준배렬로 초기화한다.
        IntList (const int A[], int n );
        IntList (const IntList &A);
        // 구축자복사
        ~IntList();//해체자
        // 대입연산자
        IntList& operator = (const IntList &A);
        // 상수목록에 있는 요소를 얻는 검토자
        int operator[](int i) const;
        // 비상수목록에 있는 요소를 얻는 검토자
        int& operator[](int i);
        // 목록의 크기를 얻는 검토자
        int size() const { return NumberValues;
        // 크기재설정함수
        void resize(int n = 0 , int val = 0);
        void push_back(int val); //새 요소를 추가한다.
    private:
        // 성원자료들
        int NumberValues;
        int* Values; //목록의 크기
        int *Values; //목록에 있는 요소에 대한 지적자
};
```

IntList정의는 3개의 구축자를 보여 준다. 첫 구축자는 기정구축자이다. 이 구축자는 지정된 요소수와 그 값을 나타나는 2개의 파라메터를 가지고 있다. 두번째 구축자는 복사구축자이다. 세번째 구축자는 존재하는 배렬로부터 IntLsit객체를 만드는 특수한 구축자이다.여기서는 이 구축자에 대하여 취급하지 않는다.

IntList구축자는 2개의 자료성원 NumberValues와 Values를 초기화한다.int자료성원 NumberValues의 목적은 목록의 크기를 얻는것이며 int*자료성원 Values의 목적은 목록안의 값을 얻는

동적기억기를 가리키기 위해서이다.

8장에서 먼저 서술한바와 같이 C++에서는 콤파일러가 복사구축자의 판번호와 클라스에 대한 성원값주기연산자를 자동적으로 만들어야 한다. 제공된 콤파일러는 매 원천자료성원을 해당대상자료성원으로 한 비트씩 정돈, 복사시키는 기능을 수행한다.이전의 클라스개발에서(실례로 Rational, Randomint) 이 판번호를 리용하였지만 여기서는 이것을 리용할수 없다. 객체가 기억기를 동적으로 얻을수 있는 고급한 클라스에서 자료성원의 앞방향에 대한 복사는 기억기모순을 가져 올수 있다.

실례로 IntList복사구축자가 없다고 하자. 이때 다음의 코드가 복사구축자로서 동작할 때 어떤 현상이 나타나는가를 보자.

```
IntList C(5.0);
IntList D(C);
```

IntList객체 C가 창조될 때 그 자료성원들인 NumberValues와 Values는 초기화된다. 구체적으로 목록에서 여러가지 요소를 구하기 위하여 동적공간이 얻어 진다. 객체 D의 자료성원 Values를 비교하는 과정이 진행되지 않는다. 왜냐하면 콤파일러 즉 제공된 복사구축자가 가정되므로 객체 C의 두 자료성원은 직접 객체 D에 복사된다. 그러므로 객체 C와 D는 자기의 Values성원과 같은 동적공간을 제공한다. 이러한 내용을 그림 11-4에서 보여 주었다. 여기로부터 C 혹은 D에 대한 변환은 다르게 된다. 실례로 C[0]을 1로 설정한다.

```
C[0]=1;
```

그때 객체 D의 현시

```
Cout << D << end1;
```

는 5개의 0으로 표시되던 초기값이 아니라 다음과 같이 출력된다.

```
[10000]
```

이러한 측면은 사용자가 자기의 목록을 관리하는데서 아무것도 기대할수 없게 한다. 그러므로 콤파일러 즉 제공된 복사구축자는 리용할수 없다.

또한 제공된 번역기성원대입연산자도 리용할수 없다. 콤파일러복사구축자와 같이 콤파일러 즉 제공된 성원대입연산자는 원천객체의 자료성원을 목적객체의 자료성원으로 직접 복사한다. 이렇게 이 성원별 복사는 역시 다른 효과를 나타낸다.

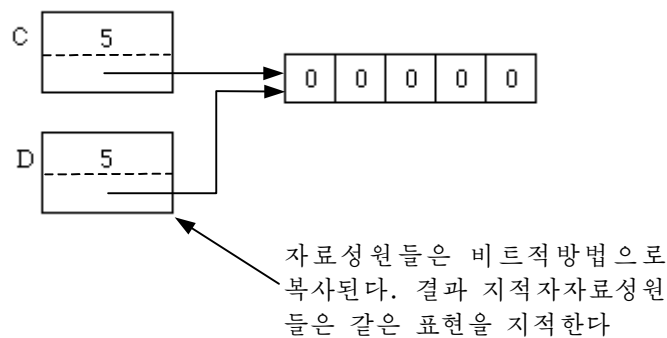



그림 11-4. 하나의 IntList에서 다른 곳으로의 성원별복사

ADT에서 다른 경우의 복사를 피하기 위하여 IntList복사구축자와 성원대입연산자를 명백히 만들어

야 한다. 두 성원은 필요한 복사동작을 수행하여야 한다. 만일 목록 11-1에서 성원대입원형과 혼돈할수 있는데 이때 값주기연산자의 오른쪽연산수만을 지정해야 한다는 8장의 내용을 참고하시오(왼쪽연산수가 일변되는 객체라는것을 리해하여야 한다). 또한 값주기연산자는 하나의 값을 돌려 주므로 아래에서 보여 주는것처럼 더 큰 값주기식을 구성할수 있다.

```
IntList A(11, 28);
IntList B;
IntList C;
C = B = A;
```

값주기연산자는 효과적으로 되돌림을 진행되며 이 방법에서는 복사가 필요없다.

IntList객체가 차지한 동적공간의 리용에서는 객체를 끝낼 때 동적공간을 돌려 주는 해체자 ~IntList()를 정의해 주어야 한다. 이것은 IntList객체가 없어 질 때 기억기루실이 없다는것을 보여 준다.

목록 11-1에서 보여 준 IntList클라스도 첨수연산자를 리용하여 두가지로 다중정의할수 있다.

```
int operator[] (int i) const ;
int& operator[] (int i);
```

성원첨자연산자의 원형은 자기의 귀환값형태와 성원함수수식자 **const**사용에서 다르다. **const**수식자는 성원함수기호의 한 부분이다. 그러므로 콤파일러가 IntList객체에서 첨수연산자의 호출을 번역할 때 어느것인가를 결정하는데 리용된다. 수식자 **const**를 리용한 첨자연산자의 정의는 **const** IntList객체가 검열되는 상태에서 호출된다.

```
const IntList A (3.5);
cout << A[2] << end1;
int i=A[1];
```

여기서 귀환값의 형태는 연산에 맞는다. 참조를 가진 첨자연산자는 형을 돌려 주며 함수수식자 **const**는 비상수형 IntList객체가 검사되거나 변경되는 상태에서는 리용된다.

```
IntList B;
B[0]=1;
Swap(B[3], B[4]);
cin >> B[5];
```

우의 실례에서 귀환형태의 참조는 맞으며 호출되는 개별적요소들은 변경될수 있다. IntList검사함수 size()의 목적은 목록에 표시된 요소의 수를 돌려 주는것이다(자료성원 NumberValues).

함수 size()는 독립적인 클라스로 정의되여 있다. 이 함수는 대체로 머리부파일에서 정의되지만 여기서는 함수의 형태를 론한다. 자기의 클라스내에서 정의되는 성원함수들은 내부전개(inline)기능에 의하여 호출된다. 내부전개기능을 제공하는 함수의 호출이 프로그람번역시에 나타나면 콤파일러가 호출되는 시점에서 본체를 대응시킨다. 이러한 대응은 성원함수의 창조와 소멸에서 품이 들게 한다. 이것은 겉으로 중요해 보이지 않지만 장기간 함수를 호출하는 프로그람에서는 아주 위험하다. 내부전개기능에 대한 품은 흔히 실행단위를 더 크게 한다.

size()함수에서 성원함수 Resize()와 push_back()는 Vector의 성원함수 Resize()와 push_back()를 가진 IntList와 비슷하다. 여기서는 Resize()와 push_back()에 대해서 취급하지 않는다.

11.9.2 IntList구축자의 실현

목록 11-2에 IntList성원함수에 대하여 정의한다. 여기서는 IntList기정복사구축자와 IntList해체자를 제공한다. IntList 기정구축자에는 2개의 파라메터 n과 val이 있다(n과 Val의 기정값은 10과 0이다). 파라메터 n은 목록요소의 초기개수를 지정한다. 파라메터 val은 이 목록요소의 초기값을 지정한다. 간단히 목록의 크기를 나타내는 assert()명령을 리용한다. 그다음 구축자는 구축되는 목록의 크기를 나타내는 자료성원 NumberValues를 설정한다.

기정구축자는 **new**연산자를 리용하여 옹근수값 n에 대한 동적기억기할당을 요구한다.

```
Values = new int[n]
```

만일 이 요구가 정확히 실행되지 않으면 례외가 주어 지지 않았을 때 프로그람을 끝낸다. Assert()는 Values의 값이 0이 아닌가를 검증한다. C++콤파일러는 기억기요구가 정확히 진행되지 않으면 Values의 값을 0으로 만들어 준다.

목록 11-2. IntList의 2개의 구축자와 해체자

```
// 구축자는 요소들을 주어 진 값으로 초기화한다.
IntList::IntList (int n, int val) {
    assert(n > 0);
    NumberValues = n;
    Values = new int [n];
    assert(Values);
    for ( int I = 0 ; I < n; ++ I) {
        Values[I] = val;
    }
}
// 복사구축자
IntList::IntList(const IntList &A) {
    NumberValues = A.size();
    Values = new int [NumberValues];
    assert(Values);
    for( int i = 0; i < A.size(); ++i)
        Values[i] = A[i];
}
// 해체자
IntList::~IntList() {
    delete[] Values;
}
```

이렇게 콤파일러가 리용되지 않았을 때 그 값을 리용하여 **for**순환이 진행된다면 Values는 목록의 요소들을 표시하기 위한 충분한 기억구역을 지정한다. 기정구축자에서 **for**순환명령은 Value가 지정하는

동적기억기의 블로크를 만드는 n개의 int객체들을 위하여 반복한다.

```
for (int i = 0; i < n; ++i) {
    Values[i] = val;
}
```

블로크의 i번째 객체는 그 값이 0부터 n-1까지인 목록의 i번째 요소와 관련된다. 배렬기호는 i번째 요소를 정확히 설정하기 위하여 지적자 Values를 리용하였다.

복사구축자와 기정구축자는 좀 류사한데가 있다. 복사구축자는 원천객체 A에 의하여 표시되는 목록의 크기를 리용하는 자료성원 NumberValues를 설정하는것으로 시작한다. 론리적으로 객체를 고찰하므로 A의 크기를 확인할 필요는 없다. 그다음 복사구축자는 A목록을 복사하기 위하여 충분한 기억구역을 요구한다. 만일 동적기억요구가 정확히 만족되면 for순환은 그 기억구역을 초기화한다.

복사구축자에서 for순환은 A에 의하여 표시된 요소의 값을 Values가 지적하는 동적기억기의 정확한 위치에 복사한다. 사실 복사를 진행하는 for순환에서의 값주기명령은 두개의 다른 첨자연산자를 리용하는데서 아주 흥미 있다.

```
Values[i] = A[i];
```

식 Values[i]는 배렬첨수연산자의 리용이며 식 A[i]는 const객체를 위한 IntList첨수연산자의 리용이다. 이것은 첨수연산이 진행되였다는것을 콤파일러에 알려 준다.

복사구축자와 값주기연산자에 의하여 진행되는 복사의 종류는 깊은 복사이다. 깊은 복사에서 분리된 대상목록은 원천목록으로부터 개별적으로 복사된 요소로 이루어 진다. 이러한 복사의 형태는 복사연산이 지적자 Values의 값을 중복하는 얕은 복사와 차이난다. 깊은 복사에서 원천과 대상목록은 동등하지만 명확히 구별된다. 원천과 대상객체의 Values자료성원은 같은 값을 가지면서도 서로 다른 기억기를 가리킨다. 얕은 복사에서 두개의 목록은 하나의 표시로 되여 있다.

11.9.3 IntList해체자의 실현

8장에서 취급한것처럼 해체자의 목적은 존재할 필요가 없는 클라스형객체에 대하여 필요한 동작을 수행하는것이다. 자료성원객체에 대하여 해체자는 더미로 동적기억기를 돌려 준다. IntList해체자는 이와 같은 과업을 수행한다. 이 함수본체는 Values가 지적하는 기억기를 해방할것을 체계에 알려 주는 명령문으로 구성된다.

```
delete[] Values;
```

이 회복은 IntList객체가 기억기를 공유하지 않기때문에 적합하다. 이러한 정확한 해체자정의가 없으면 기억기루실은 IntList객체가 없어 질 때마다 발생한다.

11.9.4 첨수의 실현

const와 비상수형 IntList객체에 다중정의된 2개의 첨수연산자를 목록 11-3에 주었다. 첨수연산자는 색인값 i가 자기의 한계내에 있는가를 검사하는 assert()마크로를 리용한다. 두개의 다중정의의 되돌림형이 다르다는것을 보기로 하자. 상수형 IntList객체의 첨수연산자의 판본은 같은 값을 돌려 준다. 값돌림은 기본요소가 변화되는것을 막는다. 비상수형객체의 첨수연산자판본은 되돌림참조이다. 왼쪽값을 만들어 되돌림값을 변경하거나 호출할수 있다.

목록 11-3. IntList.cpp의 IntList첨수연산자

```
// 개별적인 상수요소의 검사
int IntList::operator[](int i) const {
    assert((i > = 0) && (i < size()));
    return Values[i];
}
//개별적인 상수형요소의 촉진자를 얻거나 변경시키는 연산자
int& IntList::operator[](int i) {
    assert ((i >=0) && (i < size()));
    return Values[i];
}
```

11.9.5 IntList성원의 값주기와 this지적자

IntList에 다시 값주기를 하려면 다음과 같은 4개의 동작을 수행하여야 한다.

- 목록의 크기를 재설정
- 존재하는 동적기억기의 되돌리기
- 새로운 동적기억기의 충분한 할당
- 동적기억기에 원천객체의 요소복사

객체 A는 오른쪽연산수로 표시되는 IntList파라메터를 가지는 방법으로 이와 같은 동작을 수행할수 있다(즉 원천객체).

```
NumberValues = a.size();
delete [] Values;
Values = new int [numberValues];
for(int i = 1; i < = A.size () ; ++i) {
    Values[i] = a[i];
}
```

우의 프로그람은 성원값주기연산자의 함수본체라고도 할수 있다. 객체 X가 IntList객체이라면 다음의 값주기는 무엇을 실행하는가?

```
X = X;
```

값주기는 정의하지 않은 객체를 만든다. 이 경우에 오른쪽, 왼쪽연산수는 같은 객체이다. 성원값주기연산자의 파라메터 A는 (X)를 호출하는 객체의 별명이다. delete명령이 객체호출에 의해 조종되는 동적기억기를 해방할 때 호출되는 객체 (X)의 자료성원 Values값은 정의되지 않는다. A때문에 X도 목록을 만들지 않은 기억기를 호출하도록 정확히 예약할수 없다.

문제가 가장된 수법으로 나타날수 있기때문에 프로그람작성자들은 이 명령을 쓰는데서 필요없다고 소홀히 하면 안된다. 실례로 문제로 되는 객체는 일부 함수들의 파라메터 또는 전역객체와 파라메터로서 여러번 입력된다.

두 연산수가 다른 객체를 표시한다면 이러한 문제를 해결하는 간단한 방법은 값주기하는것이다.

C++는 이러한 값주기를 할수 있는 호출객체에 대한 지적자를 제공한다. 지적자는 예약어 this에 의하여 참조된다. this의 값을 &A와 비교할수 있는데 이 값은 A의 위치이다. 만일 위치가 서로 다르다면 호출객체는 변경되여야 한다.

성원값주기연산자는 목록 11-4에서 완전히 정의하였다. 이 프로그람을 보면 성원값주기연산자는 호출객체와 A가 서로 다른 객체인가를 검사하는것으로 시작한다. 만일 같지 않다면 호출객체는 갱신된다. 값주기연산자는 또한 새 표현식이 요소들과 다른 수인가를 검사한다. 그렇다면 현재의 동적기억기는 해방되며 새로운 동적기억기가 얻어 진다.

목록 11-4. IntList.cpp에 있는 IntList성원함수들과 보조함수들, 연산자들

```
// 대입
IntList& IntList::operator = (const IntList &A) {
    if (this != &A) {
        if (size() != A.size()) {
            delete [] Values;
            NumberValues = A.size();
            Values = new int [A.size()];
            assert(Values);
        }
        for (int i = 0; i < A.size(); ++i)
            Values[i] = A[i];
    }
    return *this
}
```

만일 새로운 표현식과 존재하는 표현식이 같은 요소수를 가진다면 새로운 표현식을 보관하기 위하여 이미 존재하는 동적객체를 사용한다. Values에 A의 요소를 복사하는 for순환명령에 의하여 새 표현식이 설정된다. 호출되는 객체가 변경되는가 하는것을 고찰하지 않고 연산을 완성하기 위하여 호출하는 객체의 현재값을 돌려 주어야 한다. this앞에 간접연산자 *를 붙여서 성원값주기연산자의 값을 정확히 돌려 줄수 있다.

11.9.6 IntList객체에서 삽입연산자의 다중정의

목록 1-5에서 삽입연산자가 다중정의되였다. 보조연산자는 괄호안에 목록을 현시한다. 목록에 있는 개별적요소들은 공백에 의하여 분리되였다.

목록 11-5. IntList.cpp에서 IntList삽입연산자

```
// auxiliary insertion operator for IntList
ostream& operator<<(ostream &sort,const IntList &A) { sort<<'[';
    // write out the elements
    for (int i=0;i<A.size();++i)
```

```
        sout<<a[i]<<" ";
    sout<<"]";
    // all done
    return sout;
}
```

이 삽입연산자는 상대적으로 정확히 동작한다. 괄호의 삽입은 개별적인 요소를 현시하는 for순환의 앞뒤에 생긴다. 다중정의연산자는 ostream의 되돌림값을 참조한다. 흐름을 바꾼다면 IntList삽입은 더 큰 삽입식의 부분으로 될수 있다.

```
IntList A (5,1);
cout << A << endl
```

IntList클라스에 대한 고찰은 이것으로 끝낸다. 련습을 통하여 IntList서고의 확장된 기능을 더 학습할수 있다.

문 제

25. 다음의 클라스선언을 분석하시오.

```
class Node {
    ...
public:
        int Value;
        Node *Next;
}
```

다음의 코드를 분석하시오.

```
Node Anode;
Node * p=new Node(25, null);
Node *q;
Node *tp;
Node NodeArray[mynodes];
```

tp가 NodeArray의 세번째 요소를 지정하는 명령을 작성하시오.

P가 지적하는 값을 q가 가리키게 하는 명령을 작성하시오.

P가 가리키는 마디의 Value마당을 100으로 설정하는 명령을 작성하시오.

ANode의 다음마당을 NULL로 만드는 명령을 작성하시오.

26. Position객체들의 배렬복사에 대한 지적자를 돌려 주는 C++함수 Copy를 작성하시오. Copy함수는 복사하여야 할 배렬과 배렬의 크기를 파라메터로 받는다.
27. 류동소수점형객체를 동적으로 할당하는 C++명령을 작성하시오.
28. 100개의 요소를 가진 옹근수배렬 Scores를 동적으로 할당하는 C++명령을 작성하시오.
29. 다음과 같은 클라스 Stack가 있다. 이 탄창은 Position객체를 가지고 있다. 탄창은 동적으로 할당된

배렬을 리용하여 실현된다.

```
# ifndef STACK_H
# define STACK_H
class Stack{
    public;
        Stack(int StackSize = 20);
        Stack(const Stack &s);
        ~Stack();
        // 촉진자
        // 탄창에 위치를 넣는다
        void Push(const Position &p);
        // 탄창에서 요소를 꺼낸다
        // 요소를 돌려 준다
        Position Pop();
        bool Empty() const;
    private:
        int MaxStackSize; // 탄창에 넣을수 있는 요소의 최대수
        int StackTop;
        Position *Values;
};
#endif
```

선언

```
Stack Example(4)
```

는 처음 4개의 요소를 가진 텅 빈 탄창을 만든다.

클라스 Sstack의 구축자를 구하시오.

클라스 Stack의 해체자를 구하시오.

클라스 Stack의 복사구축자를 구하시오.

11.10 알아 둘 점

✓ 왼쪽값은 평가될수도 있고 변경될수도 있는 객체를 표현하는 식이다.

✓ 오른쪽값은 평가만 할수 있는 식이다.

✓ 지적자는 다른 객체의 위치를 나타내는 객체이다.

✓ 매형의 객체에 대하여 각이한 지적자형이 존재한다. 그의 객체들이 다른 지적자들에 대한 지적자로 되는 7개의 지적자형이 있다.

✓ 객체의 위치는 주소연산자 &를 리용하여 구할수 있다.

✓ 문자 0은 임의의 지적자객체에 할당할수 있다. 문자 0은 빈 주소를 의미한다.

✓ 현재위치의 객체값은 그 위치에 대한 참조해제연산자 *를 리용하여 구할수 있다. 참조해제연산자는

왼쪽값을 만든다.

- ✓ 빈 주소는 참조할수 없는 위치값이다.
- ✓ 간접성원선택연산자 ->는 지적자가 지적하는 객체의 성원을 참조한다.
- ✓ 증가감소연산자는 지적자객체에 대하여 정의된다.지적자연산자는 갈기연산자와 관계연산자를 리용하여 비교된다.
- ✓ 지적자가 파라메터로 쓰일 때 참조해제연산자를 리용하여 참조파라메터를 모의할수 있다.

콤퓨터의 력사

마이크로쏘프트의 창설

파울 알렌과 빌 게이츠는 씨애틀에서 고등학교를 다니면서 콤퓨터를 공부하는 과정에 서로 알게 된 친구들이다.그들은 서로 콤퓨터에 깊은 흥미를 가지고 있었다. 게이츠와 알렌은 다른 동무들과 함께 프로그람을 작성하는 조직을 무었다. 고등학교시절에 그들은 어떤 급수 있는 기관의 교통운수자료를 분석하여 20000$를 벌었다.

에드워드 로버트의 Altair 8800이 1974년에 잡지《Popular Electronics》에 출현하였을 때 알렌은 보스턴에서 프로그람작성자로 일하였고 게이츠는 하바드에서 수학을 전공하고 있었다. 알렌은 이 기사를 읽고 즉시에 그것이 아주 의의 있다는것을 알았다. Altair는 새 세대콤퓨터 즉 개인용콤퓨터를 예고하였다. 알렌은 즉시에 하바드에 가서 게이츠를 만났다. 그는 아주 좋은 계기가 왔다고 말하였다. Altair는 사용할수 있는 프로그람이 있어야 하였으며 알렌은 Basic해석기를 작성하기로 결심했다. Basic는 아주 인기 있는 프로그람이였다. 게이츠는 앨부쿼쿠에 있는 로버트를 찾아 가서 자기와 알렌이 Altair에 탑재될수 있는 Basic해석기를 개발하겠다고 말하였다. 로버트는 이미 다른 사람들에게서 그러한 청구를 받았지만 Basic해석기를 내놓겠다고 한 사람은 당신이 처음이라고 게이츠에게 말하였다.

게이츠와 알렌은 Altair에서 실행되게 될 Basic해석기를 작성하기 시작했다. 그러나 그들은 Altair가 없는 상태에서 프로그람을 개발하여야 하였다. 그들이 이런 조건에서 어떻게 프로그람을 개발하였는가? 그들은 8800과 같은 동작을 하는 하바드의 콤퓨터로 프로그람을 작성하였다. 그다음 BASIC를 검사하기 위하여 이 프로그람을 사용할수 있었다. 게이츠가 Basic로 작업하는 동안 알렌은 모의계산기로 작업하였다.

2월말까지 그들은 적어도 8주정도 작업하였으며 로버트에게 이 프로그람을 보여 주었다. 알렌은 로버트에게 가서 연구사업을 하면서 Basic해석기를 계속 갱신하였다. 게이츠는 하바드에서 대학 2학년생활을 끝 마쳤다. 그리고 앨버쿼쿠에서 알렌과 합동연구를 시작하였다. 그들은 1975년 Microsoft(이음표는 후에 빼버렸다.)라는 조합을 만들었다. 로버트, 알렌, 게이츠는 여러 곳을 다니면서 이 Basic가 실행되는 Altair를 소개하였다. 마이크로쏘프트는 Basic로 첫 시발을 떼였다. 그리고 다른 회사의 콤퓨터들에서 실행될 프로그람을 작성하기 시작하였다. 마이크로쏘프트는 1980년에 큰 충격을 받게 되였다. 그때 IBM도 개인용콤퓨터영업을 시작하기로 결정하였다. IBM은 자기들의 콤퓨터(PC라고 한다.)에서 실행되게 될 쏘프트웨어개발을 외부에 맡기기로 결심하였다. IBM은 마이크로쏘프트에 IBM-PC에서 실행될수 있는 조작체계와 Basic해석기를 만들어 줄것을 부탁하였다.

IBM-PC는 대중용으로 급속히 전파되였으며 마이크로쏘프트도 급속히 장성하였다. 1985년 마이크로쏘프트는 조작체계로서 Windows를 내놓았으며 그것은 소개되자마자 많은 사람들에게 팔리였다. 현재 마이크로쏘프트는 PC와 애플의 Macintosh콤퓨터에서 동작하는 프로그람에서 론박할수 없을 정도로 큰 위력을

파시하고 있다.

게이츠와 알렌은 대부호이며 그들은 각기 수십억딸라를 가지고 있다(게이츠는 지금 세계적으로 가장 큰 부자로 되였다). 알렌은 호드킨병을 가지고 있다는 진단을 받고 1983년에 마이크로쏘프트를 나왔다. 그는 후에 회복되여 쏘프트웨어를 개발하는 여러개의 작은 회사를 만들고 거기에 투자하는 사람으로 되였다. 빌 게이츠는 마이크로쏘프트의 사장이며 21세기의 콤퓨터를 구성하고 개발하는데서 중요한 역할을 하고 있다.

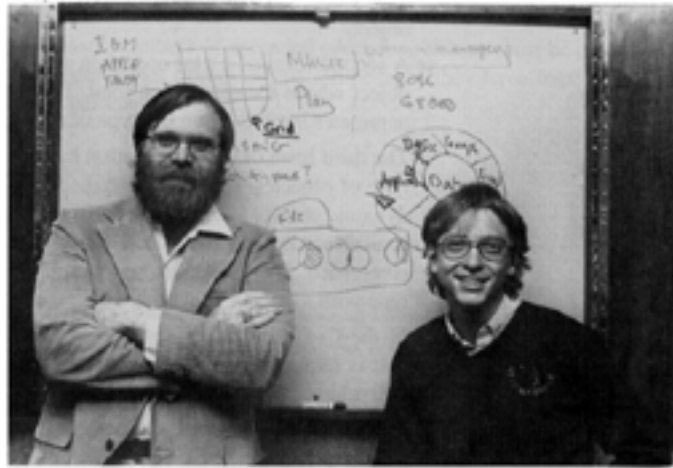



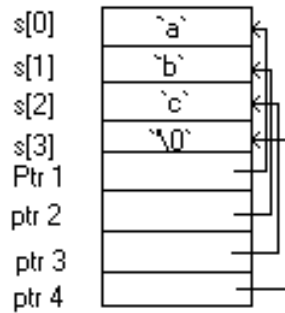
그림 11-5. 1975년도의 파울 알렌과 빌 게이츠

- ✓ 배렬이름은 C++에서 상수형지적자로 나타난다. 이 사실은 목록에서 값을 호출 및 변경하는데 리용하는 표기에서 유연하다는것을 말해 준다.
- ✓ 지령행파라메터들은 지적자를 리용하여 프로그람에 전달된다.
- ✓ 지령행파라메터를 호출하기 위하여 함수 main()은 2가지 형식파라메터를 가지고 있어야 한다. 첫 파라메터는 int형인데 그 값은 지령에 대한 실제적인 파라메터수보다 하나 더 큰 수로 자동적으로 초기화된다. 두번째 형식파라메터는 char*형의 배렬이다. 배렬의 매개 요소는 문자렬에 대한 지적자이다. 첫 요소는 지령 그자체를 가리킨다. 다른 배렬요소들은 지령에서 주어 진 여러가지 실제적파라메터들을 가리킨다.
- ✓ 습관적으로 main()의 첫 형식파라메터는 이름을 argc로 하며 두번째 형식파라메터는 argv로 한다.
- ✓ 함수에 대한 지적자객체들을 정의할수 있다. 이러한 객체들은 대체로 함수파라메터로 리용된다. 이 파라메터형은 과제수행에 더 유연하게 리용되는 함수를 제공한다.
- ✓ 동적객체는 기억기에 대한 구체적인 요구결과로서 프로그람실행과정에 창조된다.
- ✓ 동적기억기는 빈 기억기로부터 생긴다고 할수 있다.
- ✓ new연산의 기본형태는 연산자에 대하여 오른쪽연산수와 같은 형을 요구한다. 충분한 빈 기억기가 주어 진 객체형으로 리용될수 있다면 하나의 동적객체의 위치를 돌려 준다. 동적객체는 그 객체의 형을 위한 기정구축자가 있는 경우에만 초기화된다.
- ✓ new연산의 두번째 형태는 위치가 귀환된 동적객체의 초기화를 지원한다. 초기화는 할당된 객체의 형에 따라 실제파라메터목록과 같이 설정된다.
- ✓ new연산의 세번째 형태는 요구되는 객체형에 따라 첨수식을 가진다. 식은 돌려 지는 동적객체의 수를 표시한다. 충분한 경우 잇닿은 빈 기억기를 쓸수 있으며 객체가 요구하는 수를 얻을수 있는 블로크에 대한 지적자가 돌려 진다. 블로크에서 객체들은 그 형이 클라스형인 경우에만 기정구축자로 초

기화된다.

- ✓ 표준C++에서는 new연산자가 요구하는 동적기억구역이 돌려 줄수 없는 경우 례외가 발생한다. 현재 일부 콤파일러들에서는 빈 주소(0)를 돌려 준다.
- ✓ 제한된 표준C++는 set_new_handler()함수를 포함한 새로운 서고를 정의한다.이 함수는 그 파라메터로서 빈 주소를 가지고 호출하는 경우 불안정한 요구에 대하여 0을 돌려 주는 new연산자를 만든다.
- ✓ 국부객체와 달리 동적객체는 그것들을 창조하는 함수를 실행한 다음에 생길수 있다. 동적객체는 delete연산자로 할당된 요구를 해제할 때까지 존재한다.
- ✓ 주어 진 new연산으로 얻은 객체는 배렬로써 해제하여야 한다.
- ✓ 예속지적자는 해제된 기억기를 지적하는 지적자객체이다.
- ✓ 기억기루실은 해제되지 않은 동적으로 얻은 기억기인데 지적자가 가리키는것은 없다.
- ✓ 동적객체사용은 ADT로 진행된다. 표시하기 위하여 ADT는 동적객체를 지적하는 많은 자료성원을 요구한다.
- ✓ 해체자는 객체가 존재를 끝 마칠 때 자동적으로 호출되는 성원함수이다.
- ✓ 동적객체를 지적하는 자료성원을 가진 클라스형에 대하여 해체자는 보통 자료성원을 가리키는 동적기억기를 돌려 주는 delete연산을 수행한다.
- ✓ 동적기억기를 리용하는 클라스객체는 일반적으로 복사구축자, 해체자, 값주기연산자를 제공하여야 한다. 다른 경우 복사구축자와 성원값주기연산자의 기정판본을 리용한다. 그러한 리용은 잘못된것이 아니다.
- ✓ 열쇠단어 this는 호출한 객체에 대한 지적자로서 성원함수 혹은 연산자본체에서 리용된다. this지적자는 대표적으로 ADT에 대한 값주기연산자의 실행에서 리용된다.
- ✓ 수식자는 성원함수기호의 부분이다. 콤파일러는 다중정의된 함수에서 const 혹은 const가 아닌 성원이 호출되는가를 식별할수 있다. const성원은 const형객체를 호출하며 const가 아닌 성원은 const가 아닌 객체를 호출한다.
- ✓ 제공된 정확성의 수를 표시하기 위하여 클라스를 개발하여야 한다. 기본형들은 그것들이 표시에서 고정된 크기를 가지므로 그렇게 할수 없다.

련습문제

11.1 지적자객체가 표시할수 있는 값형태들은 무엇인가? 설명하시오.
11.2 왼쪽값이란 무엇인가? 오른쪽값이란 무엇인가?
11.3 오른쪽값은 왼쪽값으로 될수 있는가, 왼쪽값은 오른쪽값으로 될수 있는가를 설명하시오.
11.4 주소연산자의 목적은 무엇인가? 참조해제연산자의 목적은 무엇인가?
11.5 형이 다른 지적자들을 값주기할수 있는가?
11.6 지적자객체가 하나 증가하면 어떻게 되는가?
11.7 지령행파라메터는 프로그람에 어떻게 전달되는가?
11.8 C++가 왜 객체의 매형에 대하여 서로 다른 지적자형을 가지는가를 추측해 보시오.
11.9 다음의 그림에 대응하는 정의를 주시오.

11.10 지적자형에 대한 문자상수는 무엇인가? 목적은 무엇인가?

11.11 배렬이름과 지적자객체는 어떻게 비슷한가? 그것들은 어떻게 다른가?

11.12 변경할수 없는 float값목록을 표시하는 파라메터 A를 선언하는 두가지 방법을 설명하시오.

11.13 Ptr는 int*형이며 i도 int형이라고 하자 . 다음의 코드토막에서 이 문제를 찾으시오.

```
Ptr = i ;
i = Ptr ;
i = &Ptr ;
Ptr =Ptr + Ptr ;
i = Ptr;
&i = *Ptr ;
```

11.14 다음의 코드토막에서 왼쪽값과 오른쪽값을 찾으시오.

```
cin << k ;
i = 0 ;
A[A[3]] = 2 * b;
```

11.15 다음의 조건에 따르는 객체를 정의하시오.

1) double객체를 지적할수 있는 지적자 P.

2) 지적자객체를 지적할수 있는 지적자 Q. 여기서 지적자객체는 char형객체를 지적하는 객체를 지적한다.

3) 두개의 int 형참조파라메터를 가지는 bool형함수를 지적할수 있는 지적자 R.

11.16 int형객체 i의 위치를 지적하도록 초기화되는 지적자 P를 정의하시오. 이 정의를 계속 P의 값으로 변경시키는것은 틀린다.

11.17 int형객체의 위치를 지적하도록 초기화하는 지적자 P를 정의하시오. 이 정의에서 P를 통하여 간접적으로 i의 값을 련속적으로 변경시키는것은 옳지 않다.

11.18 다음의 코드토막의 결과는 무엇인가? 설명하시오.

```
char A[ ] = "Patricia";
char *Ptr = A;
++ Ptr ;
cout << Ptr << end1;
Ptr += 2 ;
cout << Ptr << end1 ;
```

```
cout <<--Ptr << end1;
```

11.19 다음의 코드토막의 결과는 무엇인가? 설명하시오.

```
char  A[] = "Rust never sleeps";
char *Ptr ;
for (Ptr = &A[16] ; Ptr! = A ; Ptr -= 2){
    cout << Ptr << end1 ;
    cout  << *Ptr <<end1 ;
}
```

11.20 다음의 코드토막에서 문법적으로 잘못된것은 무엇인가? 설명하시오.

```
char A[] = "James";
char B[] = "Gertrude";
char *Ptr1 =  & A [ 4 ] ;
char *Ptr 2 =  & B [ 5 ] ;
if (Ptr1 < Ptr2)
    cout << "Hello "<< end1;
else
    cout << "Good bye" << end1;
```

11.21 동적객체의 유효범위는 무엇인가?

11.22 new연산의 3가지 형태를 서술하고 매 형태의 실례를 드시오.

11.23 delete연산의 2가지 형태를 서술하시오. 이것들의 리용은 언제 적합한가?

11.24 빈 기억은 무엇인가?

11.25 기억기루실은 무엇인가?

11.26 다음의 지령을 실행해 보시오.

```
C:> play A - flat four beats
```

또한 play지령이 다음과 같이 시작되는 main()함수의 정의를 생각해 보시오.

```
int main(int argc,char* argv[ ]) { // ...
```

1) rgc 는 무엇을 표시하는가?

2) argv [0] 은 무엇을 표시하는가?

3) rgv [1] 은 무엇을 표시하는가?

4) argv[argc -1]은 무엇을 표시하는가?

11.27 다음의 정의를 분석하시오.

```
char *cPtr;
char *sPtr[12];
char **cPtrPtr;
```

1) cPtr는 char형인가? 설명하시오.

2) sPtr는 12개 요소를 가진 Char형배렬에 대한 지적자인가? 설명하시오.

3) 다음의 값주기가 성립하는가? 설명하시오.

```
cPtrPtr =&cPtr;
```

11.28 다음의 정의를 분석하시오.

```
Rational A[10];
Rational B[10];
Rational C;
Rational *D;
```

다음의 값주기가 성립하는가? 설명하시오.

```
B =A ;
C = A [ 1 ] ;
D = A ;
D = &C ;
```

11.29 다음의 프로그람토막을 분석하시오.

```
void f(int *iPtr) {
    * iPtr = 1;
}
int main() {
    int i = 0 ;
    f(&i) ;
    cout  << i << endl;
    return 0;
}
```

1) 이 프로그람은 문법적으로 정확한가?

2) 함수 f()의 정의로부터 main()함수에서 f(&i)호출은 참조파라메터에 대한 실험으로 되는가?

11.30 다음의 토막은 무엇을 의미하는가?

```
cout << **Ptr << endl;
```

11.31 프로그람에 대한 마지막파라메터는 입력된 파일의 이름이다. main()이 대면부 **int** main(**int** argc, **char** *argv[])를 가지는 경우에 main()함수에서 흐름변수(stream variable)를 아래와 같이 정의하고 초기화하는 코드부분을 해석하시오.

1) ifstream myin (argv[argc-1]);

2) ifstream myin (argv[argc]);

3) fstream myin (argc[argr]);

4) ifstream myin (argv[0]);

11.32 다음의 프로그람은 무엇을 출력하는가?

```
#include <iostream>
```

```
#include <string>
using namespace  std;
int main() {
    char *s[5]={"BASIC","IS","EAT","NAG","ENTER"};
    char **sPtr[5] = { &s[4],&s[3],&s[0],&s[2],&s[1] };
    char ***sPtrPtr=&sPtr[1];
    cout << &s[0][4];
    cout << (*((*sPtrPtr-2))+1) << endl;
    cout << *(s+1) <<  endl;
    cout << &s[3][2];
    cout << (**sPtr)[4];
    cout << *(s+2) << endl;
    return 0;
}
```

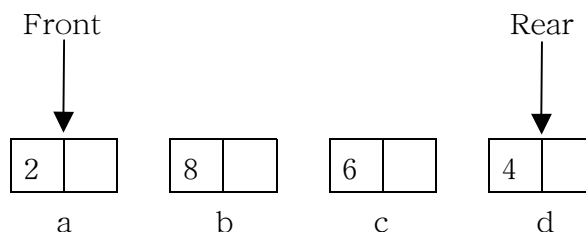
11.33 3개의 파라메터를 요구하는 void형함수 Sort()를 정의하시오. 첫 파라메터는 int형배렬 A이며 두번째 파라메터는 옹근수형값 n이며 세번째 파라메터 LTE는 두개의 옹근수형파라메터를 요구하는 bool형함수이다. 비교를 진행하기 위하여 함수 sort()는 LTE를 리용하여 목록 A를 정돈한다. 호출될 때 LTE파라메터는 비교방법에 따라 참을 돌려 주며 첫 파라메터값은 두번째 파라메터값보다 작거나 같다.

11.34 두개의 자료성원 Value와 NextItem을 가지는 클라스 Item을 정의하시오. 여기서 Value는 int형이며 NextItem은 Item*형이다. 기정구축자는 Value를 0으로, NextItem을 빈 주소로 초기화한다. 알맞는 다른 구축자, 검토자, 변이자를 정의하시오.

11.35 예속지적자는 무엇인가?

11.36 열쇠단어 this는 무엇을 표시하는가? 식 *this는 무엇을 표시하는가?

11.37 set_new_handler(0)호출의 목적은 무엇인가?

11.38 4개의 Item객체 a, b, c, d를 정의하기 위하여 련습 11.34부터 Item클라스를 리용하는 코드토막을 구하시오. 코드토막은 두개의 Item*객체 Front와 Rear를 정의하여야 한다. 이 객체의 값은 다음의 표시와 일치하여야 한다.

11.39 다음의 코드토막에서 틀린것을 찾으시오.

```
char *p=new('a');
char *q=new char('b');
char *r=new char('c');
char *s=new char[100]('d');
```

```
char *t=q;
char *u=r;
delete q;
*t='e';
```

11.40 100개의 관계형객체의 동적목록을 설정하는 코드토막을 작성하시오. 매 동적객체는 0 또는 1로 표시된다.

11.41 100개의 관계형객체의 동적목록을 설정하는 코드토막을 작성하시오. 매 동적객체는 3 또는 4로 표시된다.

11.42 다음의 두개의 프로그람을 실행하시오. C++실행에서 서로 다르게 수행된 리유를 판단하시오.

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    long int counter=0;
    int *Ptr;
    do {
        Ptr=new int;
        ++counter;
        } while(Ptr);
    cout << counter << endl;
    return 0;
}
#include <iostream>
#include<string>
using namespace std;
int main(){
    long int counter=0;
    int *Ptr;
    do {
         Ptr= new int[I];
         ++counter;
         }while (Ptr)
    cout << counter<<endl;
    return 0l
}
```

11.43 체계에서 할당할수 있는 빈 기억기의 최대블로크를 결정하는 실험을 하시오. 결과를 표시하시오.

11.44 다음의 코드토막은 무한순환을 표시하는가? 왜 그런가?

```
while (true) {
```

```
        int *p = new int;
        if (p == 0){
        break;
        }
        delete  p;
    }
```

11.45 문자렬표를 표시하기 위하여 클라스를 설계하시오.클라스는 적어도 두개의 구축자를 가진다. 기정구축자는 10개를 기입할수 있는 표를 창조한다. 다른 구축자는 지정된 표에서 기입하는 수를 결정하여야 한다. 기본검토자함수는 표에 문자렬이 표시되는가 하는것과 어떤 문자렬이 표시되는가를 가리킨다. 기본변환함수는 표에 문자렬을 추가하거나 소거하는것이다. 클라스에서 복사구축자, 해체자, 성원값주기를 정의하는것이 필요한가? 표가 다 차서 다른 문자렬을 추가하려면 무엇을 하여야 하는가?

11.46 두개의 파라메터배렬 A와 n개 요소수를 요구하는 IntList구축자를 위한 기구를 정의하시오. 구축자는 n개 요소를 가진 IntList실례를 만든다.이 요소들의 초기값은 A[0], A[1],…, A[n-1]로 주어 진다.

11.47 IntList성원함수 push_back()를 위한 기구를 정의하시오. 함수는 하나의 파라메터 val를 가진다. 함수는 현재목록의 마지막에 요소를 추가하기 위하여 IntList객체의 식을 갱신한다. 새 요소의 값은 val이다.

11.48 다음의 코드토막에서 수행되는 첨자연산의 여러가지 형태를 식별하시오.

```
    int A[100];
    IntList B(100,1);
    Vector<IntList> C(100);
    IntList    D[100];
    A[0]=1;
    B[0]=2;
    C[0]=B;
    C[0][1]=3;
    D[0][1]=A[0];
    D[0][1]=4;
```

11.49 IntList객체를 위하여 추출연산자를 더 넣으시오. 연산자는 IntList의 요소를 재설정하는데 편리한 값을 끌어 내려고 한다.

11.50 반복자 begin()과 end()를 포함하는 IntList서고를 위한 클라스정의를 확장하시오. 반복자의 귀환값이 **int***인 반복자를 작성하시오. 그런데 Values의 값은 begin()에 의하여 돌려 지며 빈 주소는 end()에 의하여 돌려 진다.

11.51 IntList구축자

```
    IntList(const int A[],int n);
```

을 위한 표본이 왜 배렬파라메터를 **const**로 하는가를 생각해 보시오.

11.52 IntList객체를 위하여 9장의 InsertionSort()함수를 더 넣으시오.

11.53 IntList객체를 위하여 9장의 QuickSort()함수를 더 넣으시오.

11.54 IntList객체를 위하여 9장의 BinarySearch()함수를 더 넣으시오.

11.55 다음의 클라스대면부를 가지는 클라스 Matrix를 실현하시오.

```
class  Matrix{
    public:
        // default  constructor
        Matrix( int r = 10, int c = 10, int v = 0 ) ;
        // copy constructor
        Matrix ( const  Matrix &M ) ;
        // destructor  operator
        ~Matrix ( );
        // assignment   operator
        Matrix &   operator = (const Matrix &M);
        // inspector for row of constant matrix
        const  IntList &  operator [ ](int i)  const;
        //inspector for row of  nonconstant  matrix
        IntList &  operator [ ] (int i);
        //inspector for size of a given row
        int  RowSize (int i) const;
        //inspector  for size  of a given column
        int ColumnSize (int i)  const;
        //inspector for number of row
        int  NumberRows (int i)   const;
    private:
        // data members
        int  NumRows;           // row size of matrix
        IntList  *Values;       // pointer to rows
}
```

- Matrix성원함수에 대하여 다음과 같이 설명할수 있다.
- Matrix(int r=10,int c=10,int v=0):은 r행과 c렬로 되여 있는 행렬을 표시하는 객체를 초기화한다. 요소의 초기값은 v이다.
- Matrix(const Matrix &M):은 M의 깊은 복사(DC:deep copy)인 객체를 초기화한다.
- ~Matrix():은 Values가 가리키는 동적기억기를 돌려 준다.
- operator=(const Matrix &M):는 M의 깊은 복사인 객체를 재설정한다.
- operator[ ](int i) const :참조는 상수형으로 행렬의 i행을 돌려 준다.
- operator[ ](int i):참조는 상수가 아닌 형으로 행렬의 i행을 돌려 준다.
- RowSize(int i) const:는 i행의 크기를 돌려 준다.
- ColumnSize(int i) const:는 i렬의 크기를 돌려 준다.

제 12 장. 검사와 오유수정

소 개

쏘프트웨어개발에서 두가지 중요한것은 검사와 오유수정이다. 검사의 목적은 완성된 쏘프트웨어를 내놓기에 앞서 임의의 조건을 가지고 검사하여 보는것이다. 쏘프트웨어검사는 쏘프트웨어를 질적으로 만들기 위한 기본부분이다. 큰 쏘프트웨어대상과제에서 검사와 오유수정은 대상과제를 완성하는데 드는 비용의 40~50%이다. 전문적인 쏘프트웨어대상과제를 위하여 쏘프트웨어회사는 매 개발자들에게 한두명이상의 검사원을 붙여 준다. 오유수정은 검사에서 나타나거나 사용자에 의하여 제기된 문제를 수정하는 과정이다. 오유수정은 문제를 분석하고 그것을 수정하는 두 단계를 거친다. 검사와 마찬가지로 오유수정도 제대로 하지 못하면 많은 시간과 자금이 들게 된다. 이 장에서는 문제가 생겼을 때 쏘프트웨어를 검사하고 오유를 수정하는 기본방법을 보게 된다.

기본개념

- 검은 통검사
- 흰 통검사
- 검사
- 단위검사
- 집중검사
- 체계검사
- 명령문적용범위
- 동등구분
- 회귀검사
- 경계조건
- 코드평가
- 검사도구
- 경로적용범위

12.1 검 사

사람들은 자기가 사용하는 프로그람에서 오유나 문제점을 찾는다. 만일 매우 긴 본문을 넣은후 문서편집기가 파괴되는 경우를 겪어 보았다면 어떻게 오유를 수정하는지 알게 된다. 일부 오유는 받은 충격이 너무 커서 신문기사로까지 실리게 된다. 1999년에 화성과 화성착륙선의 충돌이 가까운 실례이다. 이 사건에서는 많은 시험에도 불구하고 발견하지 못한 문제때문에 1억 6,500만딸라의 손해를 보았다. 착륙선이 왜 충돌하였는가 하는 이야기는 완전히 검사하기가 어렵다는것을 보여 주었다.

착륙과정은 다음과 같이 가정되였다. 착륙선이 화성의 대기속에 들어 갔을 때 락하산이 펴진다. 표면에 일정한 정도로 가까와 지면 락하산을 떼여 버리며 착륙선의 세 다리가 착륙위치에 고정되고 착륙선의 12개 엔진은 비행기가 안전하게 착륙할수 있도록 불을 내뿜어 속도를 떨군다. 착륙기의 3개 다리에는 다리가 표면에 닿을 때 비행기의 착륙엔진을 멈추게 하는 신호를 보내는 수감부가 있다.

같은 착륙선을 리용하여 연구사들은 다리들이 땅에 고정될 때 다리수감부가 진동에 의하여 허위신호를 보내게 된다는것을 확인하였다. 이러한 각본에 따라 엔진은 비행기가 거의 130피드높이에 있을 때 멎었으며 착륙선은 시간당 50마일의 속도로 떨어 졌다.

어떻게 이 문제가 발견되지 못했는가? 여러가지 검토를 통하여 개별적인 체계들의 시험에는 문제가 없다는것이 증명되였다. 즉 전반적인 체계시험에 문제가 있다는 결론에 도달하였다. 그러나 수감부는 그 검사에서 제대로 응답하였으며 결과 그 어떤 문제도 검출되지 않았다. 이후에는 전반적인 체계시험을 예산과 시간의 부족으로 하여 진행하지 못하였다.

화성착륙선의 실례는 완전한 시험이 어째서 그리도 어려운가를 보여 준다. 개별적으로 검사하면(단위검사) 모든 구성요소는 정확히 동작하지만 전체로서 검사하면(체계검사) 체계는 정확히 동작하지 않는다. 즉 체계검사를 모두 해야 할뿐아니라 단위검사도 충분히 하여야 한다. 체계의 일부분이 변하지만 자금과 시간의 부족으로 하여 앞으로 시험이 어렵게 될것이라는것을 자주 느끼게 된다. 세심한 프로그람작성자들은 자그마한 변화가 생겨도 쏘프트웨어를 다시 검사한다.

프로그람을 작성한후에 프로그람이 정확히 동작한다는것을 어떻게 확인하는가? 그리 세심하지 못한 프로그람작성자들은 두세번정도 입력한 값에 대한 결과가 정확한가를 본다. 아주 간단한 프로그람에서는 이것이면 충분하지만 아주 복잡한 프로그람에 대해서는 충분하지 못하다.

이 장에서는 설계하고 실행시키는 쏘프트웨어를 검사하기 위한 여러가지 방법에 대하여 보게 된다. 아쉽게도 검사에 대한 충분한 실례는 이 책에 없다. 쏘프트웨어검사에 대한 리론, 파학 그리고 기술을 쓴 유명한 책들이 많다. 12.4에서는 검사방법과 수속에 대하여 도움이 되는 두세가지 리론을 준다.

검사가 고급쏘프트웨어를 만드는데서 만병통치약이 아니라는것을 참고하여야 한다. 콤퓨터파학자 에드가 딕스트라의 유명한 말이 있다. 그는 《프로그람검사에서는 오유가 존재한다는것은 보여 줄수 있지만 오유가 전혀 없다는것은 절대로 보여 줄수 없다.》고 하였다. 고급쏘프트웨어는 다른 쏘프트웨어공학기술을 리용하여 시험할수 있다. 현실검사와 가상검사가 필요하다. 여기에는 실지 코드뿐아니라 쏘프트웨어의 기능, 체계의 설계나 구조에 대한 검사가 있어야 한다. 사실 쏘프트웨어공학을 배우는 사람에게는 검사보다도 프로그람을 규칙대로 작성하는것이 오유를 피하는데서 더 효과적이다. 또 한가지 중요한것은 개발되는 쏘프트웨어를 효과적으로 관리하고 추적하기 위한 능력이다. 오유추적을 위한 원천코드조종체계와 쏘프트웨어는 일반적으로 이러한 과제들을 수행하기 위하여 리용된다.

12.1.1 검사실례

오유에 대한 검사에서는 문제들을 빨리 찾으면 찾을수록 더욱 좋다. 쏘프트웨어공학을 학습하면 문제를 발견하고 수정하는 원가가 시간에 따라 기하급수적으로 높아 진다는것을 알수 있다. 실례를 들어 코드작성시 오유를 찾으면 비용이 거의나 들지 않는다. 가령 오유를 수정하는데 1딸라가 든다고 하자. 그와 같은 오유를 쏘프트웨어검사의 마지막단계에서 발견하였다면 그것을 수정하는데 수백 혹은 수천딸라가 든다.

이것은 코드를 작성할 때 코드를 검사해야 한다는것을 의미한다. 이런 현상은 많이 찾아 볼수 있다. 더 큰 체계의 한 부분인 어떤 함수를 설계하고 작성한다고 하자. 이때 세부적인 문제에 부닥치게 된다. 이것은 모듈을 검사하는데서 좋다. 만일 이때 문제가 생기면 코드를 즉시에 수정할수 있다. 그러나 여러달후에 문제가 생길 때 문제를 분석하고 수정하려면 코드를 다시 보아야 한다. 즉 함수나 모듈을 여러번 거쳐야 하는데 오유를 다시 찾는다는것은 더 어려워 질것이다. 하나의 모듈이나 함수를 검사하는 과정을 단위검사라고 한다.

검사과정 특히 단위검사를 위한 실례로서 지나간 시간을 표시하는 EzWindow LED시계프로그람을 작성하자. LED시계에는 수자와 시간을 표시하기 위하여 초단파발생기, 수자시계, 시계라지오 그리고 VCR(그날의 시간, 지나간 시간 등)와 같은 전자장치가 있다. LED시간계수기는 시간제한을 가지는 유

희를 작성하고 콤퓨터화된 득점판을 개발하는데 편리한 클라스이다.

객체지향설계방법에 따라 LED의 속성과 동작을 결정하여야 한다. 명백히 값을 조종하여야 한다. 시계와 박자발생기와 같은 다른 객체를 만들기 위하여 클라스 LED를 사용하므로 현시창문에서 LED객체의 위치를 설정하여야 한다. 또한 LED객체가 현시될 창문을 정의하여야 한다. LED를 표시하는 방법은 수를 표시하는 수자의 비트매프를 사용하는것이다. 실례를 들어 수 3의 비트매프화상은 아래와 같다.

수 0부터 9까지의 비트매프를 보기로 하자. LED클라스는 다음의 속성을 가진다.

- MyDigits:수자 0 부터 9까지의 화상들을 보유하는 비트매프들의 배렬
- MyValue: LED가 현시될 때 현시하여야 할 기호
- MyPosition:창문에서 LED의 위치.
- MyWindow:LED 가 표시되는 EzWindow

공동대면부를 위하여 매 속성의 검토자와 변이자가 있어야 한다. 추가적으로 LED를 표시하는 조종자가 요구된다. 클라스 LED의 초기선언은

```
class LED{
    public:
        LED( );
        // Inspectors
        Position GetPosition() const;
        int GetValue() const;
        // Mutators
        void SetPosition (const  Position  &p);
        void SetValue (int  d);
        void SetWindow(SimpleWindow  *w);
        //Facilitators
        void Show();
    private:
        SimpleWindow *MyWindow;
        BitMap MyPosition;
        int MyValue;
};
```

이다. 미리 클라스선언을 하였으므로 그의 실현부만 보면 된다. 실현부는 간단하며 목록 12-1에 주어 져 있다. 이제 우리는 박자발생기를 위한 코드개발을 계속 진행하든가 아니면 단위검사의 방법으로 그것이 정확히 동작하는가를 확인하기 위하여 클라스 LED를 정지시키고 검사해야 한다.

목록 12-1.　　　　　　　　　　클라스 LED의 실현부

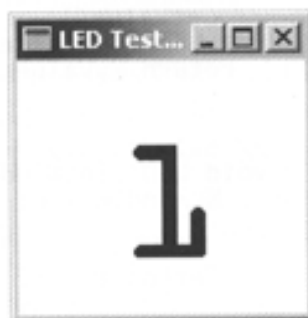
```
#include "led.h"
const int MaxDigits = 10;
char *DigitNames[MaxDigits] = {
    "digit0.bmp",
    "digit1.bmp",
    "digit2.bmp",
    "digit3.bmp",
    "digit4.bmp",
    "digit5.bmp",
    "digit6.bmp",
    "digit7.bmp",
    "digit8.bmp",
    "digit9.bmp"
};
//LED() -- [0-9]비트도형을 읽는다.
LED::LED(){
    for (int D =0;D <MaxDigits; ++D)
        MyDigits[D].Load(DigitNames[D]);
}
//GetPosition() --LED의 현재위치를 되돌린다.
Position LED::GetPosition() const{
    return MyValue;
}
//GetValue() --LED의 현재값을 되돌린다.
int LED::GetValue() const {
    return MyValue;
}
//SetWindow() --LED를 표시하기 위한 창설정
void LED::SetWindow(SimpleWindow *w) {
    MyWindow = w;
}
//SetPosition() --LED의 위치를 설정한다.
void LED::SetPosition(const Position &p) {
    MyPosition = p;
}
//SetValue() --LED의 값을 설정한다.
void LED::SetValue(int v) {
    MyValue = v;
```

```
}
//Show() --창에 LED를 표시한다.
void LED::Show() {
    MyDigits[MyValue].SetWindow(*MyWindow);
    MyDigits[MyValue].SetPosition(MyPosition);
    MyDigits[MyValue].Draw();
}
```

앞에서 언급한것처럼 문제가 나타나기를 기다리기보다 코드로 되여 있을 때 문제를 검사하고 찾는것이 더 쉽다. 그런데 채 작성하지 못한 객체를 어떻게 검사하는가? 코드를 동작시키기 위한 검사도구 및 검사토막을 만들어야 한다. 검사도구는 개발중에 있는 코드를 검사 혹은 동작시켜 보기 위하여 작성한 작은 코드부분이다.

검사도구를 쓰는것은 단위검사의 표준단계이다. 검사가 끝난후 이 코드부분을 버리기는 아쉽지만 이 부분을 보관하는데 걸리는 시간과 노력은 오유가 없을 때 보상된다. 오유를 지적하고 그것들을 없애는데 도움이 되는 검사기구들은 이미 있다. 대체로 대상과제를 위하여 리용하고 있는 여러가지 검사기구들을 보관한 검사등록부를 만든다. 다음의 코드는 LED클라스를 위한 검사기구이다.

```
#include "led.h"
SimpleWindows *W;
LED L;
int ApiMain() {
    W=new SimpleWindow("LED Test Window",4.0f,4.0f);
    Position p(1.0, 0.5);
    W->Open();
    L.SetWindow(w);
    L.SetValue(1);
    L.SetPosition(p);
    L.Show();
    return 0;
}
```

이 코드를 콤파일하고 실행하면 다음의 창문이 나타난다.

코드가 동작했다! 계획대로 검사는 정확히 되였다. 그러나 실용응용프로그람에서 사용된 코드에서는

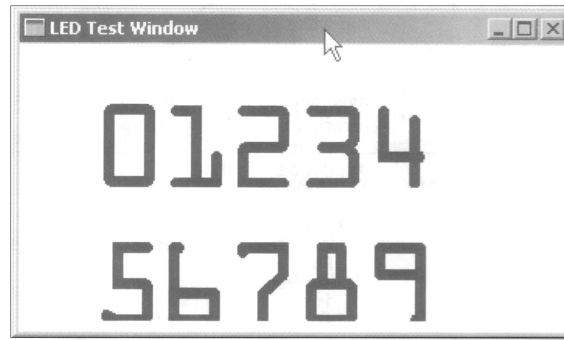
검사가 명백하지 않다. 이 코드를 완전히 검사하기 위하여 사용된 코드와 잘못된것이 무엇인가를 생각하여야 한다. 필요한것은 단위검사를 위한 체계적인 방법이다.

첫째로 코드가 요구를 만족시킨다는것을 증명하여야 한다. 클라스 LED의 요구는 무엇인가? 공교롭게도 이것들이 실지 작성되지 못하였다. 실제적인 쏘프트웨어대상과제에서 첫 단계는 클라스가 해야 하는것에 대한 형식적인 명세부를 작성하는것이다. 그러나 우리는 클라스 LED가 제공하게 될 능력에 대해서 대체적으로 알고 있다.

- 클라스 LED는 EzWindow에서 수자 0부터 9까지 표현할수 있다.
- 클라스 LED의 위치는 EzWindow에서 적당히 정해질수 있다.

이러한 가상적인 명세부로부터 검사모임을 더 광범히 설계할수 있다. 기본적으로 나타난 요구를 클라스가 만족시키도록 하여야 한다. 검사부분은 매 수자를 정확히 표시할수 있게 하여야 한다. 또한 수자표시위치를 확인하기 위한 검사도 포함된다. 간단한 작업을 통하여 한번의 실행에서 이 모든것을 진행하는 검사도구를 작성할수 있다. 위치의 정확성을 얻기 위하여 여러번 시험하고 오유를 바로 잡은 다음 검사기구는 매 수자를 표시한다.

```
#include "led.h"
SimpleWindow *W;
LED L;
int ApiMain(){
    W = new SimpleWindow("LED Test Window",8.0f,4.0f);
    W->Open();
    Position p(1.0, 0.5);
    L.SetWindow(W);
    int i;
    for(i = 0;i <5; ++i) {
        L.SetValue(i);
        L.SetPosition(p);
        L.Show();
        p = p + Position(1.0, 0.0);
    }
    p = Position(1.0, 2.5);
    for(i = 5; i< 10; ++i) {
        L.SetValue(i);
        L.SetPosition(p);
        L.Show();
        p = p + Position(1.0, 0.0);
    }
    return 0;
}
```

검사기구를 실행시키면 다음과 같이 표시된다.

결과는 LED클라스가 모든 수자를 표시할수 있으며 비트매프의 위치도 정확하다는것을 보여 준다.

세심하지 못한 프로그람작성자들은 이것으로 그친다. 그러나 세심한 프로그람작성자들은 객체가 요구에 맞지 않는 방법을 사용하면 어떻게 되는가를 또다시 검사한다. 이러한 처리를 하는 리유중의 하나는 단위검사에서 오유가 검사부분을 실행하기전에 발견되게 하자는것이다. 실례를 들어 틀린 값이 들어갈 때 객체가 그에 맞게 동작하는 검사부분에 대하여 생각한다면 이 부분을 전혀 처리하지 못하였다는것을 즉시에 알수 있다. 클라스 LED는 0보다 작고 9보다 큰 값이 통과하면 동작하지 않는다. 이것을 수정하는것이 중요하다. 즉 후에 수정하는것보다 이제 수정하는것이 훨씬 더 쉽다.

성원함수 SetValue()를 수정하여 통과한 값이 주어 진 범위가 아니라면 오유를 알려 주도록 한다. 수정된 성원함수는

```
void LED::SetValue(int v) {
    assert(v >= 0 && v <= 9);
    MyValue=v;
}
```

이다. 오유발견코드가 동작하게 하는 2개의 검사부분을 만든다. 한 검사부분은 0보다 작은 값을 통과시키고 다른 검사부분은 9 보다 큰 값을 통과시킨다. 검사할 때 모든 검사부분을 다 통과하도록 검사를 진행한다.

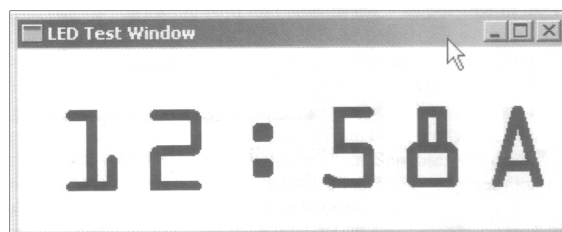
이제는 검사가 만족스럽게 진행되였다. 그러나 아직 끝나지 않았다. 클라스 LED는 박자를 만든다. 이 응용프로그람에서 클라스 LED가 동작하는가를 검사하기 위하여 시계모형을 먼저 현시할수 있다. 원래 정해 진 시계대면부가 만들어 져 있는데 시간은 변하지 않는다. 이 동작은 클라스 LED가 여러가지 기능을 놓치고 있다는것을 보여 준다. 12:30P 혹은 11:00A와 같이 시간을 표시하기 위하여 클라스 LED는 오전시간과 오후시간을 각각 표시할수 있도록 두점과 문자 A, P를 표시하기 위한 능력이 요구된다.

이제 이러한 특징들을 포함하여 클라스 LED의 실행을 수정한다. 시계대면부를 만들기 위한 코드를 작성하기 위하여 LED객체의 길이와 너비를 얻을수 있다면 집합객체의 위치지정에 편리하다. 해당한 BitMap함수를 호출하여 LED를 표시하는데 리용되는 비트매프의 길이와 너비를 얻을수 있으므로 실행이 간단하다. 이것은 대면부에서 오유나 문제를 찾는 한가지 실례이다. 이 오유나 문제를 찾는것은 프로그람오유를 찾는것만큼 중요하다. 목록 12-2에 시계표본을 위한 검사기구코드를 주었다.

목록 12-2. 시계모형검사기구

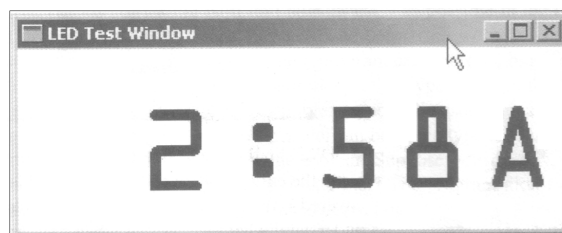
```
#include "led.h"
SimpleWindow *W;
```

```
LED Clock[6];
int ApiMain() {
    W = new SimpleWindow("LED Test Window",8.0f,2.5f);
      Position p (0.5,0.5);
    W->Open();
    int i;
    for (i = 0; i < 6; ++i) {
        Clock[i].SetWindow(W);
        Clock[i].SetPosition(p);
        p = p + Position(Clock[i].GetWidth(),0.0);
    }
    //시간을 표시한다
    Clock[0].SetValue(1);
    Clock[1].SetValue(2);
    Clock[2].SetValue(Colon);
    Clock[3].SetValue(5);
    Clock[4].SetValue(8);
    Clock[5].SetValue(AMIndicator);
    for(i = 0; i < 6; ++i) {
        Clock[i].Show();
    }
    return 0;
}
```

검사기구에 의하여 나타나는 화면은 아래와 같다.

표본단위검사처리는 클라스 LED와 함께 또 다른 문제가 있다는것을 보여 준다. 1:30P와 같은 시간을 어떻게 표시하는가? 공백을 표시하는 방법이 요구된다(즉 어두운 LED). 더우기 이 새로운 기능을 추가하는것은 클라스 LED와 매우 류사한것만큼 이 상태에서는 쉽다. 새로운 기능을 추가한 다음에 추가된 기능을 검사하는 또 다른 하나의 검사프로그람을 첨부한다. 마지막으로 추가한 검사프로그람은 아래와 같다.

```
#include "led.h"
SimpleWindow *W;
LED Clock[6];
```

```
int ApiMain() {
    W = new SimpleWindow("LED Test Window",8.0f,2.5f);
    Position p (0.5,0.5);
    W->Open();
    int i;
    for (i = 0; i < 6; ++i) {
        Clock[i].SetWindow(W);
        Clock[i].SetPosition(p);
        p = p + Position(Clock[i].GetWidth(),0.0);
    }
    //시간을 표시한다.
    Clock[0].SetValue(1);
    Clock[1].SetValue(2);
    Clock[2].SetValue(Colon);
    Clock[3].SetValue(5);
    Clock[4].SetValue(8);
    Clock[5].SetValue(AMIndicator);
    for(i = 0; i < 6; ++i) {
        Clock[i].Show();
    }
    return 0;
}
```

이 검사프로그람은 다음과 같은 화면을 만드는데 공백을 만들수 있는가를 확인한다.

클라스 LED를 변화시킬 때마다 검사프로그람전체를 다시 실행해야 한다는데 주의를 돌려야 한다. 이것은 여러개를 변화시킨 다음에 오유가 발생되는것을 피할수 있다. 하나만 변화되였을 때 무엇이 잘못되였는가를 훨씬 쉽게 알수 있다.

클라스 LED에 대한 검사를 진행하면 검사에서 왜 시간이 걸리며 그것이 왜 쏘프트웨어개발에서 중요한 부분으로 되는가를 알수 있다. 매 변경후 전반적인 검사를 진행하는것은 시간을 소비하는 일이지만 앞으로의 품을 적게 들이게 한다. 프로그람작성자들은 대체로 자동적으로 검사하고 오유를 알려 주는 스크립트를 리용한다. 스크립트를 사용하면 검사는 명령을 호출하는것만큼 간단하다. 새로운 검사프로그람이 작성되는 경우 이 검사프로그람을 스크립트에 넣는다. 이 스크립트들에는 앞으로 다른 개발자들이 검사프로그람의 동작방법을 알게 하기 위하여 검사프로그람을 넣는다. 이렇게 스크립트는 앞으로 개발자들과 검사자들에게 문서로서 봉사된다.

문 제

1. 대체로 대상과제의 몇프로가 검사와 오유수정에 소비되는가?
2. 검사와 오유수정사이의 차이점을 설명하시오.
3. 단위검사란 무엇인가?
4. 검사기구란 무엇인가?
5. 목록 9-8에 주어 진 함수 Checkword()를 위한 검사기구와 검사부분을 만드시오.
6. 공백, A, P 그리고 두점을 포함한 모든 수자를 표시하는 검사기구를 수정하시오.

12.1.2 검사원리

앞에서 언급한바와 같이 검사는 질이 높은 쏘프트웨어를 만드는데 필요하다. 검사는 자체의 기술용어와 연구결과를 가진 부분이다. 검사의 목적은 개발과정에 될수록 빨리 오유를 찾고 쏘프트웨어를 수정하는것이다. 정확히 오유란 무엇인가? 사실 프로그람이 폭주된다면(즉 푸른 화면) 그것은 오유이다. 그러나 여러가지로 명백치 않은 오유들도 많다. 실례를 들어 문서편집기에서 단어를 강조하기 위하여 도구띠에서 bold단추를 눌렀을 때 선택된 단어에 변화가 없다고 생각해 보자. 이것이 오유인가? 프로그람은 끝나지 않았지만 지령대로 동작하지 않았다. 이것도 역시 오유이다.

음향쏘프트웨어공학기술을 실례로 하여 이 쏘프트웨어가 진행하는것에 대한 완전하고 세부화된 명세부와 그것의 동작방법 그리고 그것이 지원하는 기능들과 지원하지 않는 기능들, 그의 성능요구에 대해서 보자. 일반적으로 오유는 프로그람에 이러한 정의가 없을 때 발생한다. 그러나 검사자와 대부분의 프로그람작성자들은 오유를 다음의 4가지 부류로 나눈다.

- 쏘프트웨어의 폭주나 자료의 변화
- 정의되여있지 않거나 정확치 않다.
- 빈약하거나 부족한 수행
- 사용하기 힘들다.

쏘프트웨어의 폭주는 프로그람에서 주목하였던 방법이 실패하는 경우 생긴다. 쏘프트웨어의 폭주실례로서 프로그람이 필요없이 탈퇴하거나 지령에 의해 정지되는것을 들수 있는데 실지로 조작체계지령이 《죽었을 때》이다.

자료변화는 프로그람이 파일에 잘못된 자료를 쓰기 하였을 때 생긴다. 흔히 쓰는 문서편집기로 파일을 편집하고 기억한후 파일에 영문 모를 단어가 있다고 생각하여 보시오. 이것은 자료변화오유실레이다. 자료변화오유는 쉽게 발견할수 없기때문에 마음을 놓을수 없다. 오유는 다른 자료파일로 전염될수 있으며 오래동안 발견되지 않으면 변화된 파일은 정확한 상태로 재생하기 어렵다.

쏘프트웨어정의에서 중요한 구성요소는 체계가 제공하는 형태이다. 형태목록은 모든 사용자들이 체계가 기능상 완성되였음을 알수 있도록 한다. 레를 들어 수산기프로그람을 작성할 때 프로그람은 여러가지 진수들사이의 변환을 지원하여야 한다. 만일 이러한 형태가 없거나 미완성이면(실례로 2진수로만 변환할수 있다.) 그때는 오유로 된다.

기능오유는 프로그람이 기능상 요구를 만족시키지 못하는 경우 생긴다. 기능오유의 한가지 실례로서 전자우편체계가 보관된 통보를 검색할수 있는 기능을 가지고 있다고 가정해 보시오. 그 기능을 가지고 작업한다면 검색이 지내 오래기때문에 이 형태를 리용할수 없다. 그것은 전자우편체계가 그러한 형태밖에 제공하지 못하기때문이다.

전형적인 사용자대면부는 쏘프트웨어를 이전보다 더 사용하기 쉽게 하여 준다. 그러나 사용자들은 쏘프트웨어의 대면부가 더욱 더 편리해 질것을 요구한다. 수준이 다른 사용자들이 쏘프트웨어설계를 쉽게 할수 있게 한다는것은 매우 어려운 일이다. 실제로 사용자대면부와 그 리용의 편리성은 콤퓨터과학의 보조적인 부분이다. 만일 프로그람설계가 과제의 요구를 만족시키기 어렵다면 이것도 오유 즉 사용자대면부설계에서의 오유이다. 참고적으로 이러한 오유는 대상과제의 완성에서 지나치게 많은 비용을 소비하게 한다. 따라서 일부 사람들이 말하듯이 프로그람사용을 쉽게 또는 친절히 하기 위한 사용자대면부검사를 빨리 하는것은 매우 중요하다.

프로그람을 검사할 때 프로그람폭주와 자료변화가 가장 흔히 보게 되는 오유이다. 이러한것은 명백히 동작상 오유이다. 한편 프로그람이 이상하다 할 정도로 속도가 떠지고 사용하기 어려워 진다면 이것도 오유인가? 개발자는 이러한 비정상적인 현상이 오유가 아니라 자체의 결함이며 프로그람의 속도가 뜨지 않고 실지로 사용하는데서 어렵지 않다고 주장할것이다.

중요한것은 검사의 한계를 리해하는것이다. 앞에서 언급한바와 같이 검사를 통하여 프로그람에 오유가 없다는것을 확인할수 없다. 단지 검사를 통하여 오유를 찾고 퇴치할수 있을뿐이다. 앞으로 일부 프로그람들에 대하여 프로그람검사를 철저히 한다는것은 거의 불가능하다. 문제는 대부분의 프로그람들에 입력하는 자료의 수가 많고 프로그람안에서 명령문들의 갈래가 복잡하다는데 있다. 이 문제를 고찰하기 위하여 프로그람안에 있는 가능한 갈래의 개수를 생각해 보시오. 실례로 조종흐름도가 그림 12-1과 같은 간단한 프로그람을 보자.

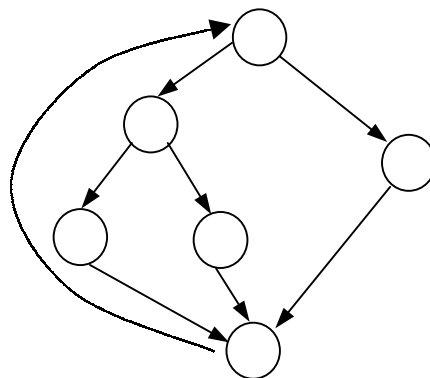



그림 12-1. 프로그람조종흐름도

매원은 명령문들의 블로크를 의미한다. 이 프로그람을 완전히 검사하기 위하여서는 프로그람의 가능한 경우들을 다 실행시켜 보아야 한다. 순환을 내놓고 프로그람에는 서로 다른 3개의 갈래가 있다. 만일 순환이 20번 실행된다면 3^{20}개의 명령문들의 조합을 실행하여야 한다. 이것은 검사하여야 할 부분이 거의 30억개나 된다는것을 의미한다. 프로그람실행의 가능한 모든 경우를 다 검사한다는것은 전혀 불가능하다. 즉 가능한 모든 오유를 찾을수 있는 효과적인 검사방법이 있어야 한다.

12.1.3 고찰과 검토

앞에서 오유를 더 빨리 찾을수록 더욱 좋다는것을 설명하였다. 설계와 코드고찰의 목적은 코드를 실행하기전에 오유를 찾자는것이다. 설계나 코드에 대한 고찰을 통하여 프로그람작성자는 다른 프로그람작성자에게 설계나 코드의 내용을 리해시킬수 있는 현실적인 처리는 물론 가상적인 처리에 대해서도 오유를 찾을수 있다. 이러한 고찰은 현실적이든, 가상적이든 개발과정의 첫 단계에서 오유를 찾을수 있게 한다. 그러나 고찰이 정확히 진행되면 다른 우점도 가지게 된다.

고찰은 프로그람에 포함된 모든것을 배우는 과정으로도 된다. 고찰과정에 사용자는 문제를 해결하기 위한 아주 섬세하면서도 효과적인 설계나 기술을 보게 된다. 또한 이 과정에 사용자는 잘못되거나 빈약한 코드부분도 보게 된다. 성공뿐아니라 오유를 통해서도 배울수 있다. 고찰은 첫 프로그람작성자들이 현실적인 요구와 코드들을 배우도록 도와 준다. 큰 대상과제에 대한 작업을 할 때 매 프로그람작성자들이 선택된 표준코드를 아는것이 중요하다. 표준코드는 프로그람설계방법을 의미한다(즉 여러가지 언어구조를 사용하는데서의 차이, 해설의 형태와 내용에 대한 요구, 허락되지 않은 구조 등). 같은 표준코드의 사용은 더 믿음직하고 더 쉬운 코드를 얻을수 있게 한다.

고찰은 또한 대상과제관리에서 효과적이다. 고찰은 대상과제성원들의 기능을 경영자들이 평가하는데 도움을 준다. 이러한 평가는 대상과제성원들에 대한 과제할당을 결정하고 효과적인 팀을 형성하는데 리용된다. 고찰은 또한 경영자가 대상과제의 발전을 평가할수 있게 해준다. 이 정보는 우선권이동, 더 좋은 원천추가, 대상과제의 일정한 부분에 대한 더 많은 시간의 할당과 같은 정확한 동작을 줄수 있다.

쏘프트웨어공학학습은 고찰이 오유발견에서 매우 효과적이라는것을 보여 준다. 큰 쏘프트웨어에 대한 학습은 고찰이 생산성을 14% 증가시키고 결함을 90% 감소시킨다는것을 보여 주었다. 또 다른 학습은 고찰이 단위검사에 비해 거의 2배나 효과적이라는것을 보여 주었다.

매우 효과적인것으로 인정받고 있는 고찰(review)과정을 검토(inspection)라고 한다. 검토는 소속된 성원이 특정한 임무를 받는 공식적인 과정이다. 검토는 1976년에 IBM에 의하여 먼저 리용되였다. 바로 그때 설계와 코드의 검토가 대체로 오유의 60%를 없애 준다는것을 보여 주었다.

검토가 고찰과 구별되는 한가지 특징은 그것이 고도로 구조화되였다는것이다. 매 관계자는 검토를 수행하는 방법과 임무가 무엇인가에 대하여 훈련을 받는다. 검토에서 한 관계자는 4가지(조정자, 검토자, 작성자 혹은 필사자)중 하나의 역할을 수행한다.

조정자: 조정자는 검토를 수행한다.조정자의 가장 중요한 임무는 검토를 적당한 속도로 진행하는것이다. 검토는 많은 문제를 판단하는데서 철저하면서도 헛되이 질질 끌지 말아야 한다. 조정자의 수행에서 중요한것은 사용자들의 리용을 친절하게 해주는것이다. 검토를 진행하는데서 추가적으로 조정자는 검토관계자들에게 검토되는 코드나 설계의 분배,검토기관과 위치표작성,검토결과알림,그리고 검토결과가 완성될 때 생기는 여러 동작항목을 확인하기 위하여 응답한다.

검토자: 검토자 및 고찰자는 설계나 코드(실례로 구성요소를 만들기 위한 코드리용, 설계수행, 검사 등)에 여러가지 흥미를 가지는 작성자와 다른 사람이다. 검토자의 임무는 임의의 잘못된 문제를 찾기 위하여 설계나 코드를 세심히 검사하는것이다. 코드의 검토는 검토모임전에 끝난다.

작성자: 코드나 설계의 작성자는 검토에서 기본역할을 한다. 작성자의 임무가 끝나면 코드는 리해하기 쉽고 오유가 없는 훌륭한 문서로 된다. 검토자가 문제를 발견하거나 코드가 실지화면에서 명백하지 않으면 동작항목은 상태를 수정하기 위하여 작성자를 호출한다. 검토자가 때때로 오유라고 생각하는것은 사실 오유가 아니다. 이 경우에 작성자는 코드가 정확치 않은 리유를 설명할수 있다.

필사자: 필사자의 역할은 발견된 모든 오유를 기록하고 발생한 동작항목의 목록을 보관하는것이다.

흥미 있게도 경영자는 검토에서 제외된다. 쏘프트웨어검토는 될수록 빨리 많은 문제를 찾는것을 목표로 하는 기술적검사이다. 경영자는 검토의 취지를 변화시킬수 있다.

검토가 고찰과 구별되는 다른 하나의 특징은 그것이 다음과 같은 5가지의 잘 정의된 단계들로 구성되여 있다는것이다.

계획작성: 계획작성단계에서는 검토해야 할 코드부분이 선택되고 조정자가 검토자들에게 과제를 할당한다. 검토자들은 고찰해야 할 코드의 각이한 부분을 할당받거나 그 코드를 어떤 견지(실례를 들어 검사가능성, 확장가능성, 성능 등)에서 코드를 고찰해야 하는가를 질문할수 있다. 검사목록은 지나간 대상과제에서 중요한것이나 문제가 생긴 실지 부분에 대하여 검토자들이 주의를 돌리도록 해준다. 검토단계에서 참가자는 검토팀에 속하여 코드를 훑는다.

개괄: 개괄에서 작성자는 검토되는 설계나 코드에 영향을 주는 대상과제의 어떤 높은 준위부분을 서술한다. 모든 대상과제관계자들이 대상과제의 이러한 측면들을 알고 있다면 개괄단계는 생략될수도 있다.

준비: 매 검토자는 혼자서 작업하면서 제공된 검사목록을 안내서로 리용하여 코드를 세심히 고찰한다. 검토자들은 코드에서 임의의 문제나 결함들을 찾고 자기들의 결과를 보고하는 검토모임에 참가한다. 참가자로 선택된 검토자는 검토모임기간에 코드나 설계를 보여 주기 위한 계획을 준비한다. 검토처리에 대한 학습은 여기서 2시간이상 더 걸리지 말아야 한다는것을 보여 주고 있다. 코드를 읽는것은 어려운 작업이며 2시간후에 검토자는 지루하게 되고 오유를 발견할수 없다.

검토모임: 검토모임에서 참가자는 코드가 무엇을 하는가를 설명하면서 코드를 한행씩 추적한다. 참가자는 코드를 읽고 설명하면서 부분코드에서의 문제를 판단하고 토의한다. 필사자는 발견된 모든 오유와 그것들과 관련된 동작항목들을 기록한다. 조정자는 검토가 합리적인 속도로 진행되고 력량이 집중되도록 해준다. 실례를 들어 문제를 어떻게 수정하겠는가를 론하는것은 모험이다. 이것은 검토의 목적이 아니다. 준비단계와 같이 검토모임은 2시간이상 더 걸리지 않는다.

검토보고: 검토모임후 조정자는 끝내야 할 작업과 누가 매 과제에 응답할수 있는가를 판단하는 통보를 준비한다. 변화의 크기에 따라 수정된 코드의 검토가 계획된다. 검토통보는 검토의 결과에 기초한 림시목록에 대한 추가나 변화를 나타낸다. 이 정보는 련속검토의 효과성을 증명할수 있다.

검토는 코드를 읽고 리해하기 위하여 구조화된 환경을 제공하므로 효과적이다. 대부분의 사람들은 코드를 찾아 읽는것을 오히려 지루해 한다. 코드를 읽을 때 코드를 대충 보고서는 코드가 무엇을 하는가를 완전히 리해하지 못한다. 검토는 사람들이 초점을 둔 능률적인 방법으로 코드를 읽도록 한다. 검토는 또한 일반문제에 대하여 재생하여 주므로 효과적이다. 앞으로 검토를 위하여 검사목록에 이 정보를 통합하는것은 련속검토의 효과성을 높여 준다.

12.1.4 검은 통과 흰 통검사

든든하고 질이 높은 쏘프트웨어를 서술하기 위한 2가지 검사방법은 검은 통과 흰 통검사이다. 이 장의 처음에 본 클라스 LED의 검사는 흰 통검사의 실례이다. 흰 통검사라는 말은 검사부분을 만들 때 코드를 "보거나" 시험할수 있다는것을 의미한다. 검은 통검사라는 말은 검사부분을 만들 때 코드를 시험할수 없다는것을 의미한다. 코드는 볼수 없는 검은 통에 숨겨 져 있다.

그것을 볼수 없다면 코드검사를 어떻게 할수 있는가? 왜 볼수 없는 코드검사를 요구하는가? 이 두 질문에 대한 대답이 있다. 검은 통검사로써 코드작업은 할수 없지만 설명문은 코드가 무엇을 호출하는가를 말해 준다. 원천코드를 호출함이 없이 입력하고 출력을 얻으며 정확한 결과를 검사할수 있다. 두번째 질문에 대한 대답은 흰 통검사가 코드에서 기본오유를 찾는다는것이다. 코드가 설명을 수행하지 않는다면 흰 통검사는 오유형태를 찾는데서 정확치 않다. 흰 통검사의 우점은 코드작업방법지식이 너무 많은 검사부분을 피하므로 더 효과적으로 검사할수 있게 한다는것이다.

검은 통과 흰 통은 흔하므로 큰 쏘프트웨어대상과제들에 대하여 둘 다 사용된다. 이 책은 프로그람

작성에 대한것이므로 흰 통검사에 대한 론의에 초점을 모은다. 그러나 론의하는것은 검은 통검사에 더 많이 적용한다.

성공의 열쇠인 효과적인 검사는 오유를 거의 정확히 찾아 내는 좋은 검사부분을 만드는것이다. 이 과제는 다 검사하는데서 큰 프로그람에 의한 입력능력이 너무 많으므로 어렵다. 따라서 가능한 오유를 찾기 위하여 효과적이며 더 작고 더 많이 관리할수 있는 검사부분의 수를 감소시키는 방법을 찾아야 한다. 불필요하거나 너무 많은 검사부분을 없애는 처리를 동등구분이라고 한다. 동등구분에 대한 기본사상은 2개의 입력이 같은 코드부분을 검사하였을 때 검사모임에서는 하나의 입력만 요구된다는것이다. 표본검사로부터 2개의 입력은 같다. 실례를 들어 수산기프로그람을 개발한다고 생각해 보자. 추가적인 조작은 지금까지 수행하였고 다른 조작을 수행하기전에 추가작업을 완전히 끝내기 위하여 검사부분을 개발한다. 검사부분은 1+2, 2+1, 0+3, 0+0으로 한다. 수산기는 이 검사부분에 대하여 정확한 답을 만든다.검사부분 1+3을 더하는것이 필요하다고 생각되는가? 아니요. 그것은 검사부분 1+3이 1+2와 같은 동등클라스에 있기때문이다. 만일 검사부분 1+2가 풀리면 검사부분 1+3도 풀린다. 더하는데서 좋은 검사부분은 -1+3이다.이 검사부분은 첫 연산수가 부수이므로 새로운 동등클라스에 있다. 새로운 동등클라스에 있는 수산기프로그람을 위하여 추가적인 검사부분을 만들라는 문제가 주어 졌다.

효과적인 검사부분을 만드는데 프로그람작성자들과 검토자들이 사용하는 여러가지 방법이 있다. 가장 일반적인 방법의 하나는 경계검사이다. 경계검사를 하는 리유는 프로그람오유가 흔히 경계에서 생기기때문이다. 앞으로 코드가 경계에서 완성되였을 때 아무데서나 정확히 동작하게 될것이다. 다시 말한다면 매달릴곳이 없는 벼랑의 변두리를 따라 걸을수 있다면 땅우에서도 걸을수 있을것이다.코드에 따라 여러가지 형태의 경계선이 있다. 순환경계, 자료경계, 능력경계가 있다. 순환경계는 순환이 시작과 끝에서 오른쪽의것으로 되는것이다. 자료경계는 허락한 값의 경계에 있는 자료를 조종할 때 코드가 오른쪽의것으로 되는것이다. 능력경계는 묶음이 차고 빈 상태를 코드가 정확히 조종하는것이다.

흰 통검사에서는 경계조건을 보는 코드를 시험할수 있다.실례를 들어 목록 12-3에 9장에서 소개한 함수 BinarySearch()을 주었다. 이 코드의 개별적인 검토는 찾는 열쇠값이 묶음의 시작이나 끝에 있다는 경계조건을 준다. 만일 코드들이 그러한 상태들로써 동작한다면 열쇠가 벡토르 A안의 아무곳에 있을 때 정확히 동작한다.

목록 12-3. 함수 BinarySearch()

```
// BinarySearch():목록 A에서 Key를 찾는 반분탐색
int BinarySearch(vector<char> &A, char Key) {
    int left = 0;
    int right = A.size() - 1;
    while(left <= right) {
        int mid = (left + right)/2;
        if (A[mid] == Key)
            return mid;
        else if (A[mid] < Key)
            left = mid + 1;
        else
            right = mid ?1;
```

```
        }
        return A.size();
    }
```

경계조건의 한가지 실례는 벡토르 A의 크기가 1 혹은 0인 때 코드가 동작하겠는가 안하겠는가 하는 것이다. 이 두가지 부분에 대한 검사가 포함된다. 이 검사부분은 변화상태를 검사한다. 변화상태는 코드의 대표적인 사용에서 일어 나지 않지만 그것들이 생기면 프로그람이 동작되지 않는다. 순환은 벡토르의 크기에 의하여 조종되므로 이 검사들은 또한 순환경계검사로서 쓰인다. 만일 벡토르 A가 비였다면 순환은 전혀 실행되지 않는다.이 코드는 벡토르 A가 크기 1을 가질 때 동작하는가? 그 크기가 0인 때는 어떤가?

경계검사의 또 다른 실례에서와 같이 6장의 가격리용도표로부터 목록 12-4에 있는 코드를 분석해 보시오. 코드의 시험은 코드가(함수 Valid()를 리용하여) 낮은 주식값은 0이상이며 높은 주식값은 낮은값이상이라는것을 보여 준다. 이것은 직접 다음의 자료경계검사부분을 보여 준다..

최저가격	최대가격
0	0
0	10
0	-1
-1	3
10	8

첫번째 검사부분은 0인 주식값이 낮은 값과 높은 값에 대하여 맞는가를 검사한다. 두번째 검사부분은 낮은 값이 0이고 높은 주식값이 0이 아닌 부분을 프로그람이 조종하는가를 검사한다. 프로그람은 2개의 검사부분에 대한 그라프를 만든다. 다음 세번째 검사부분은 오유통보를 만들기 위한 프로그람을 생성한다. 세번째 검사부분은 낮은 주식값이 0이고 높은 주식값이 부수일 때 프로그람이 오유통보를 알리는가를 검사한다. 네번째 검사부분은 낮은 주식값의 더 낮은 경계를 검사한다. 이 검사는 하나의 오유를 발생한다.

목록 12-4. **주식도표프로그람코드**

```
//Valid(): 주별 주권가격 확인
bool Valid(float low, float high) {
    return (0 <= low) && (low <= high);
}
//ReadStockinterval():주별 최저, 최대주권가격을 읽기
bool ReadStockinterval(istream &fin,
  const string &FileName,int &Low, int &High, int Week) {
    if (fin == cin)
        cout<< "Enter the low and high stock price";
    fin >> Low >> High;
```

```
    //더이상 자료가 없다면 거짓을 되돌린다.
    if (! fin)
        return false;
    //자료가 유효한가 검사한다.
    if(!Valid(Low,High)) {
        cerr << FileName << ": Bad data for week"
          << Week + 1 << endl;
        exit(1);
    }
    return true;
}
```

검사부분은 2개의 동등클라스로 구분될수 있다. 첫 2개의 검사부분은 정확한 입력인데 마지막 3개는 정확하지 않으며 프로그람이 오유통보를 발생하게 한다. 만일 프로그람이 하나의 오유통보를 발생하지 않으면 그때는 오유가 나타난다. 일반적으로 정확히 입력하고 프로그람이 정확히 출력하였는가 아니면 잘못 입력하고 프로그람이 오유통보를 하나 발생하였는가 하는 검사부분으로 나눌수 있다.

검사부분을 만드는 또 다른 방법은 적어도 한번 실행되게 하기 위하여 매 명령문을 표시하는 검사부분모임을 만드는것이다.이것을 명령문 혹은 코드범위검사라고 한다. 기본사상은 적어도 한번씩 코드의 매행을 실행하지 않는다면 코드를 완전히 검사할수 없다는것이다. 물론 명령문범위검사는 오유를 가지는 특수한 명령문을 련속 실행하지 않으므로 오유를 놓칠수 있다. 또한 복잡한 프로그람에 대하여 완성된 코드범위를 담보해야 할 검사부분의 수는 매우 크다.

코드범위를 검사하기 위하여 프로그람명령문의 실행을 보여 주는 코드와 생성통보를 리용할수 있는 쏘프트웨어도구가 있다.

검사모임을 만들기 위한 다른 방법들도 있다.한가지 방법은 실행하는 프로그람의 조종흐름도의 매 정점을 만드는 검사모임을 만드는것이다. 이 기술을 경로범위 혹은 경로검사라고 한다. 경로검사에 대해서는 다음의 코드부분을 분석해 보시오.

```
if (x != y)
    y = 5;
else
    z = z-z;
if (x > 1)
    z = z / x;
else
    z = 0;
```

이 프로그람의 조종흐름도를 그림 12-2에 주었다. 매 변두리를 선회하게 하는 검사모임은 <X=0, Z=1>와 <X=3,Z=3>이다. 첫번째 검사부분은 실행되는 A, B, G, H경로를 만든다. 두번째 검사부분은 실행경로 E, F, C, D를 만든다. 이 검사모임과 함께 문제는 중요한 부분을 놓친것이다. 검사부분 <X=0, Z=3>이 실행되면 무엇이 생기는가? 이 문제를 풀기 위하여 모든 가능한 경로를 검사하여야 하는

데 이미 본것은 실행할수 없다.

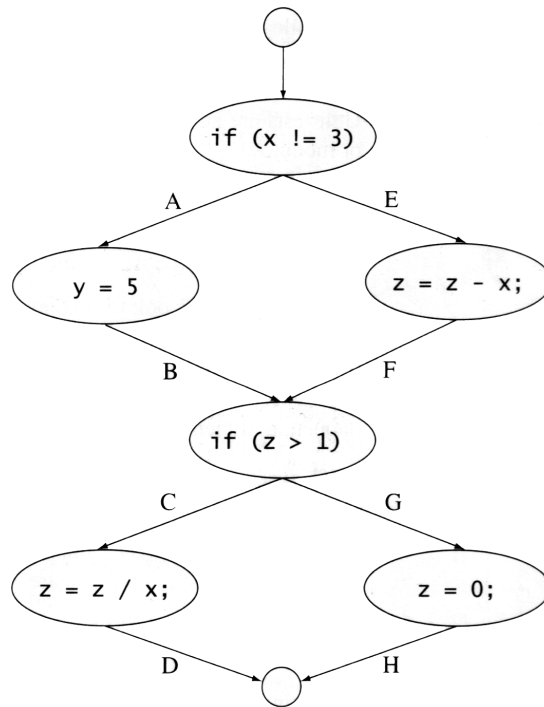



그림 12-2. 2개의 if-else명령문의 조종흐름도

검사부분을 만드는데 사용되는 많은 검사와 방법에 상관없이 검사의 중요한 구성요소는 자동이다. 쏘프트웨어가 개발되였으므로 검사는 정기적으로 다시 하여야 한다. 따라서 검사진행, 출력얻기, 실지출력과 요구한 출력의 비교를 위한 자동수속의 설정은 중요하다. 이것은 정기적으로 귀환검사를 하게 한다. 귀환검사는 쏘프트웨어의 새 판본조작을 이전 판본의 조작과 비교한다. 결과는 프로그람동작이 예견치 않은대로 변하지 않는다는것이다.

만일 한번 진행한 동작을 더 이상 하지 않는다면 복귀된다. 복귀검사는 새 오유를 알리지 않거나 낡은 오유를 반복하는것을 막는다.

자동검사의 다른 리유는 검사수속이 검사를 어떻게 하는가를 문서로 제공하는것이다. 이것은 쏘프트웨어를 공개한후 오유가 생기고 여러해 수정했을 때 도움이 된다. 자동검사수속이 없이 모든 검사를 진행하는 방법과 통과에 대한 추측, 실패에 대한 추측을 생각하여야 한다.

경험

검사경험

효과적인 검사를 위한 여러가지 일반적인 경험들이 있다.

빠른 검사

오유를 인차 찾을수록 수정하기가 더 쉽다. 또한 코드개발과정에 검사부분을 발생시키는데서도 더 쉽고 효과적이다. 단위검사는 개별적인 구성요소들이나 모듈들에서 오유를 빨리 찾기 위한 효과적인 방법이다.

검토리용

검토는 쏘프트웨어에서 오유와 부족한것을 찾는데서 매우 효과적이다. 공식적인 검토체계가 련습은 아니지만 다른 프로그람작성자들에게 코드를 설명하는데서 능률적이다.

경계검사

코드에서 경계조건을 찾고 경계와 그 주위에서 검사하는 검사부분을 만드시오. 하나의 오유를

없애는것은 보통이며 그러므로 그것들을 명확히 검사하는데 유리하다.

례외조건검사

실행되지 않는 상태에 대하여 고찰하고 그때 그에 대한 검사를 진행하시오. 대표적인 상태는 빈 파일, 자료입력, 틀린 자료, 매우 작은 자료 그리고 매우 많은 자료이다. 완성된 프로그람은 이 부분들을 모두 조종한다.

검사의 반복

검사를 진행하고 실지의 출력을 요구한 출력과 비교하는 자동수속을 설정하시오. 스크립트작성언어와 쉘언어가 이것을 위하여 리용된다.

12.1.5 종합과 체계검사

단위검사는 하나의 함수, 모듈이나 구성요소에 초점을 둔다. 부분쏘프트웨어와 함께 진행하는 검사를 종합검사라고 한다. 체계검사는 전체 체계와 함께 있을 때 하는 검사이다. 완전한 단위검사는 종합과 체계검사를 한다. 그러한 부분작업은 명백하므로 전문가들이 부분이나 구성요소들사이의 대면부검사에 초점을 모을수 있다. 앞으로 개별적인 부분작업을 갱신하기때문에 검사가 실패하면 전문가들은 구성요소들사이의 대면부에서 문제를 찾는데 초점을 모을수 있다. 좋은 단위검사를 하기 위한 기준은 종합과 체계검사에 리용된다. 차이점이 초점으로 된다. 종합검사에서 초점은 쏘프트웨어구성요소들사이의 호상작용을 검사하는데 있다.

따라서 개발하는 입력검사는 일부 체계를 동작시키는데 초점을 둔다. 이와 같이 체계검사와 함께 입력검사는 모든 체계의 동작을 검사해 보며 개별적인 구성요소들의 동작에 대해서는 하지 않는다. 그것은 단위검사에 의하여 진행된다.

검사를 위한 모듈화방법은 구성요소들이 다른 구성요소들과 최종적인 체계를 만들기 위하여 조립되여 있으므로 필요하며 전체 체계검사는 진행할수 없다.

문 제

7. 검토에서 리용되는 4가지 임무를 말해 보시오.
8. 대부분의 검토방법과정에서 참가자는 코드의 작성자와 다른 사람이다. 이것이 왜 좋은 방법으로 되는가.
9. 검은 통검사와 흰 통검사의 차이점을 설명하시오.
10. 수산기의 더하기연산을 검사하기 위한 새로운 동등클라스를 만드시오.
11. 명령문범위검사는 무엇인가?
12. 경로범위검사는 무엇인가?
13. 2진검색기능을 위한 검사기구를 설치하고 완전히 검사하시오. 발견한 오류를 말하시오.
14. 적어도 한번 실행하는 다음의 프로그람에서 매 명령문에 대한 검사부분을 만드시오.

```
int Euclid(int x, int y){
    while (x != y) {
        if (x > y)
            x = x - y;
        else
            x = y - x;
```

```
    }
    return 0;
}
```

12.2 오유수정

검사는 오유의 존재를 발견하기 위한 과정이다. 오유수정은 무슨 오유가 있는가를 표시하고 그것을 없애는 과정이다. 검사부분이 오유를 찾을 때는 흔히 오유의 리유가 명백하다. 그것들은 간단한 오유이다. 프로그람이 왜 정확히 동작하지 못하는가를 찾는것은 지루하며 숙련되지 못한 방법이 사용되면 특히 시간소비가 많다. 12.2.1에서는 과학적인 방법에 기초한 오유수정방법을 준다. 12.2.2에 프로그람작성자들이 체험한 오유수정방법에 대한 안내와 정보를 준다.

12.2.1 과학적인 방법

쏘프트웨어를 주거나 보내기전에 찾기 힘든 마지막오유를 찾는것은 헛될수도 있고 중요한 경험이기도 하므로 문제가 생긴 코드를 수정하는데서 다른 프로그람작성자에게 의뢰할 때가 많다. 이것은 하나의 실무적인 문제가 아니다. 이 절에서는 오유수정을 위한 과학적인 방법을 소개한다.

과학적인 방법은 귀납적론리에 기초한 최종적인 체계적방법이다. 과학적인 방법은 다음의 단계들을 리용한다.

자료모으기: 자료에서 실례를 관찰하고 표본을 찾는다.

가설세우기: 그럴듯한 설명을 표시하거나 검토한 구체례를 표시한다. 이것이 가설이다.

새로운 실례를 예언: 가설을 리용하여 아직 검토하지 못한 새로운 구체례나 동작을 예언한다.

동작수행: 새로운 구체례를 검토하기 위한 동작을 수행한다. 동작을 수행하고 자료를 수집한다.

가설의 증명 혹은 론박: 예언한 구체례를 고찰하면 가설이 실지로 세워 진다. 가설이 가능성을 가지지 못하면 처리는 다른 가설세우기로써 반복된다. 다른 가설을 세우기 위하여 자료를 더 수집하여야 한다.

여기에 오유수정을 위한 과학적인 방법의 리용을 례증한 간단한 실례를 주었다. 프로그람은 산수연산명령문에서 0에 의한 나누기를 없애 버린다. 잘못된 명령문에서 나눔수로 리용된 값은 묶음에서 0아닌 수의 개수를 세는 순환에 의하여 고찰된다. 이 고찰에 따라 묶음은 0아닌 값을 가지지 말아야 한다. 이 가설로부터 순환전에 묶음내용을 표시하는 코드를 추가한다면 출력은 모두 0이 될것이다. 정의된 코드의 실행을 검사한다. 만일 묶음이 0만을 포함한다는것을 보여 준다면 가설은 맞는다. 출력이 0아닌 값을 가지면 동작결과는 실지가설과 맞지 않으며 왜 나눔수가 0인가를 설명하기 위하여 또 다른 가설을 세워야 한다.

이러한 처리는 시간을 소비하지만 실지는 그런것이 아니다.흔히 가설을 검사하는 동작은 오유수정자를 리용하여 진행할수 있다.이전 실례에서 코드의 추가와 프로그람의 재콤파일을 하지 않고 순환전에 중지점을 설정하고 묶음을 표시하는 오유수정자를 사용할수 있다.

중요한것은 무엇을 발견한다는 명백한 생각이 없이 코드와 변하지 않는 명령문을 무조건 검토해야 한다.

이것은 오유수정에 리용된 과학적인 방법의 힘을 보여 주는 실례이다.셀리 코드와 추크 해커는 콤퓨터지도작성프로그람을 만들고 있는 프로그람작성자들이다. 프로그람은 위치를 가진 경계표가 들어 있는

파일을 읽고 지도를 만든다.

샐리와 추크에게 지도우에 집을 표시하기 위하여 집아이콘을 만들데 대한 초기과제가 주어 졌다고 하자. 그림 12-3 은 그리려고 하는 미완성품을 보여 준다. 그들은 붉은 4 각형을 그려 집을 표시하고 그 안에 흰 4 각형을 배치하여 창문을 표시하기로 하였다. 지붕은 푸른 3 각형으로 하였다. 그것은 EzWindows 형, RectangleShape, SquareShape 그리고 TriangleShape 를 리용하여 수행된다.

그림 12-3. 집아이콘의 대략그림

목록 12-5에서는 HouseIcon클라스에 대하여 서술하고 목록 12-6에서는 HouseIcon의 부분수행에 대하여 보여 준다. 이러한 부분작업을 한 다음 샐리와 추크는 더 많은 형태를 추가하려고 하였다.

샐리와 추크가 훌륭한 프로그람작성자라는 또 다른 의미는 HouseIcon의 초기판본을 만들고 실행한 후인데 클라스 HouseIcon이 정확히 동작하도록 하기 위한 검사기구를 리용하였다. 만일 HouseIcon에 어떤 문제가 있다면 오유수정은 이때 수행이 더 복잡한것보다는 낫다. 목록12-7에 검사기구를 주었다.

목록 12-5. HouseIcon 의 정의

```
#ifndef HOUSEICON_H
#define HOUSEICON_H
#include "ezwin.h"
#include "position.h"
#include "wobject.h"
#include "square.h"
#include "rect.h"
#include "triangle.h"
class HouseIcon : public WindowObject{
    public:
        HouseIcon(SimpleWindow& w, const Position& p);
        void Draw();
    private:
        RectangleShape HouseBase;
        SquareShape Window1,Window2,Window3,Window4;
```

```
        TriangleShape Roof;
        color HouseColor;
        Position HouseBasePosition;
        Position Window1Position;
        Position Window2Position;
        Position Window3Position;
        Position Window4Position;
};
#endif
```

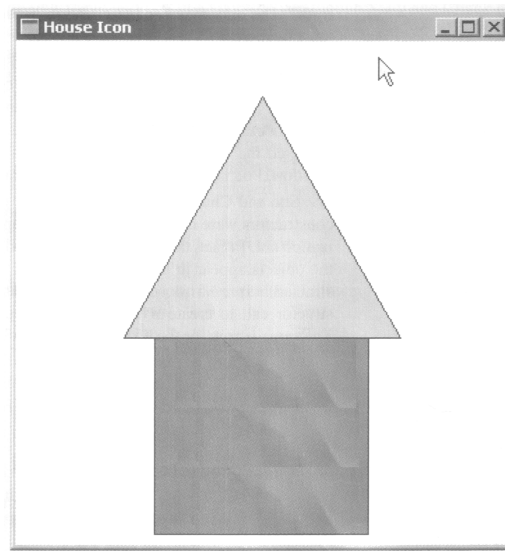
목록 12-6. HouseIcon의 부분적인 실현

```
#include "HouseIcon.h"
HouseIcon::HouseIcon(SimpleWindow& w, const Position& p):
    WindowObject(w, p),
    HouseBasePosition(p), HouseBase(w, p, Red,3.5, 4.0),
    Roof(w, p + Position(0.0, - 2.5), Green, 4.5f),
    Window1Position(p + Position(-0.5, -0.5)),
    Window2Position(p + Position(0.5, -0.5)),
    Window3Position(p + Position(-0.5, 0.5)),
    Window4Position(p + Position(0.5, 0.5)),
    Window1(w, Window1Position, White, 0.5),
    Window2(w, Window2Position, White, 0.5),
    Window3(w, Window3Position, White, 0.5),
    Window4(w, Window4Position, White, 0.5),
    HouseColor(Red) {
    //코드 필요없음
}
void HouseIcon::Draw() {
    HouseBase.Draw();
    Roof.Draw();
    Window1.Draw();
    Window2.Draw();
    Window3.Draw();
    Window4.Draw();
}
```

목록 12-7. HouseIcon의 검사기구

```
#include "HouseIcon.h"
SimpleWindow *W;
int ApiMain() {
    W = new SimpleWindow("House Icon ",10.0f,8.0f);
```

```
    W-> Open();
    Position ButtonPosition(5.0f,6.0f);
    HouseIcon HomeButton(*W,ButtonPosition);
    HomeButton.Draw();
    return 0;
}
```

놀랍게도 샐리와 추크가 프로그람을 수행하여 다음의 결과를 얻었다.

창문이 나타나지 않았다. 간단한 오유가 있다. 사실 샐리와 추크의 첫 생각은 창문을 집을 표시하는 붉은 4각형주위에 그렸다는것이다. 그런데 Draw()성원함수에 대한 검토는 창문을 마지막에 그리고 있다는것이다. 샐리와 추크는 오유를 찾기 위하여 과학적인 방법을 쓰기로 한다. 추크는 다음의 가설을 세웠다. 그는 집의 창문은 그려지지만 정확한 위치에 그려 지지 않는다고 생각하였다. 집의 창문을 그리면 계속 보일것이라고 하였다. 샐리는 EzWindows객체가 검은 판으로 되여 있으며 현시창문의 아무데나 그리면 계속 보일것이라고 하였다. 샐리와 추크는 코드를 동작시키지는 않았지만 여전히 시험을 하였다. 이 과정에 샐리와 추크는 ≪상상시험≫을 진행하였다. 맞는 상상시험을 하는것은 검사를 진행하고 코드를 실행하는것보다 훨씬 더 빠르다.

샐리와 추크는 있을수 있는 가설을 세울수 있는 더 많은 자료를 요구하였다. 샐리는 조종대로 집창문의 위치를 표시하기로 하고 그리기를 상상해 보았다. 그들은 집의 창문들중 하나를 자기 위치에 그리기로 하였다. 이것은 집의 창문들이 없이 표시되게 하였다. 그들은 함수 HouseIcon::Draw()에 다음의 명령문을 추가하였다.

```
cout<<"Window 1 position is("<<x<<","<<y<<")"
  <<endl;
```

프로그람을 실행할 때 창문에서 다음의 출력을 얻었다.

```
Window 1 position  is (-1.07374e +008,-1.07374e +008)
```

x와 y자리표는 범위바깥이다. 이것은 집의 창문이 왜 나타나지 않는가를 보여 주지만 이때 질문은 왜 자리표가 틀리는가 하는것이다.

집창문을 표시하는데 사용되는 SquareShape의 위치는 HouseIcon의 구조체로서 설정되였다. 코드는 정확해 보였다. 샐리와 추크는 난처하였다. 그들은 생각밖이였다. 그것은 오유수정때문에 시간을 소비하였기때문이다. 코드를 주의깊게 시험하여도 문제가 보이지 않는데 이것은 보통 프로그람조작이 잘못되였다는것을 의미한다. 이 경우에 프로그람의 량을 보지 않고 하는것은 명백하다. 잘못된 부분이 보이든가 문제가 없는 정확한 부분을 보게 된다. 이 상태에서 프로그람작성자가 할것은 두가지이다. 한가지 방법은 여러 사람들에게 문제를 설명하는것이다.

다른 방법은 프로그람조작에서 결함을 적게 하기 위하여 더 많은 자료를 시험하고 모으는것이다. 이 경우 프로그람조작을 확인하기 위하여 구체적인 자료가 요구된다. 이 자료를 모으기 위하여 프로그람작성자는 오유수정기를 사용하거나 프로그람을 리해하도록 해주는 묶음명령문을 추가한다.

샐리와 추크는 Window1의 위치를 표시하는 HouseIcon구조체에 cout명령문을 넣는다.

자리표는 이 점을 혼란시킨다. 자리표가 자료성원초기화목록과 구축자본체에서 설정되여 있다면 다른 코드들이 그것들사이에 실행되지 않기때문에 매우 나쁘다. 이 점에서 미숙한 프로그람작성자는 콤파일러로 잘못한 결과를 얻는다. 이것은 매우 드문 경우이다. 앞으로 일부 콤파일러로 할수 있다는 구체적인 담보는 없다. 유일하게 남은것은 자료성원초기화목록이다. 문제는 거기에 있다. 샐리와 추크는 오유수정기를 사용하며 집창문을 포함하는 매 구축자 Position과 SquareShape에 정지점을 설정한다. 아마 이 자료는 문제를 보게 해줄것이다. 프로그람이 기동할 때 SquareShape구조체가 먼저 호출된다는것을 알수 있다. Window1Position객체를 만들기 위하여 먼저 호출되는 구축자 Position을 요구하기때문에 그것은 림시적이다. 샐리는 문제가 무엇인가를 깨달았을 때 좋아 했다.

그와 추크는 자료성원초기목록에 표시한 순서대로 여러가지 구축자가 호출된다는 가설에 기초하여 코드를 작성하였다. 실지로 자료성원을 위한 구축자는 클라스설명에서 객체를 표시하는 순서로 호출된다. 객체 Window1은 Window1Position이 창조되고 초기화되기전에 구축된다. 때문에 Window1을 만들기 위한 구축자호출은 초기화되지 않은 Position을 통과한다. Window2, Window3 그리고 Window4도 같다. 샐리의 가설이 맞다면 한가지 가능한 방법은 클라스 HouseIcon의 정의에서 개별적인 자료성원들의 내용을 변화시키는것이다. 4개의 집창문위치는 먼저 나타나야 한다. 수정된 선언은

```
class HouseIcon : public WindowObject{
    public:
        houseIcon(SimpleWindow& w, const Position& p);
        void Draw();
        void MoveAbsolute(const Position& p);
        void MoveRelative(const Position& p);
    private:
        RectangleShape HouseBase;
        color Housecolor;
        TriangleShape Roof
        Position HouseBasePosition;
        Position Window1Position;
        Position Window2Position;
        Position Window3Position;
        Position Window4Position;
```

```
            SquareShape Window1, Window2, Window3, Window4;
    };
```

이다. 프로그람은 그림 12-4에 보여 준 화상을 만든다. 마침내 집처럼 보인다.

이 련습은 섈리와 추크가 자기들의 코드에 대하여 생각해 보게 한다. 집만들기는 간단히 되였다. 무엇이 잘못되였는가? 그들의 코드는 잊어 버리거나 스쳐 버리기 쉬운 C++의 알기 어려운 동작에 의존하였다. 훌륭한 프로그람작성자들은 오유로부터 배운다. 섈리와 추크는 코드가 너무나 불안하므로 어떤 사람이 그것을 확장하거나 수정하는 경우 파괴가 적어 지게 하기 위하여 HouseIcon을 수정하기로 하였다.

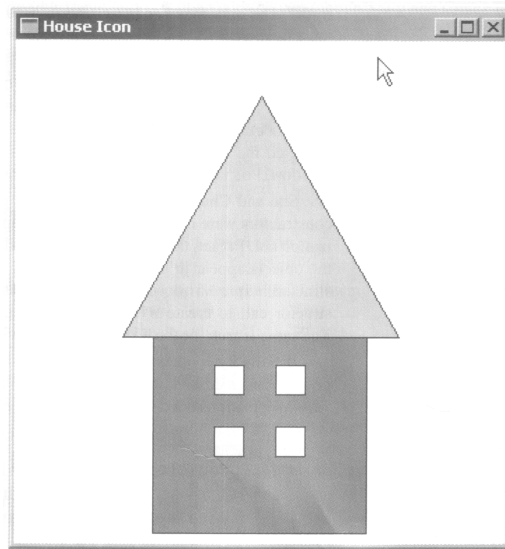



그림 12-4. 창문에서 집아이콘의 현시

12.2.2 오유수정과 그 기법

오유를 수정하는 기능은 프로그람의 착오를 막기 위하여 강력한 기능으로 된다. 오유를 없애고 나타난 오유에 대해서 제때에 진단하기 위한 여러가지 방법들이 있다. 이러한 방법들은 과학적인 방법들에 기초하고 있다.

문제의 간단화: 나타난 오유코드들을 다시 검토하자. 나타난 오유코드들을 지워 버리고 필요 없는 함수들을 삭제해 볼수 있다. 이렇게 하면 오유를 수정하려는 코드의 복잡성을 감소시킬수 있다. 오유가 나타난 코드들을 삭제해 버리고 아직도 오유가 있는가를 검사하기 위하여 프로그람을 주기적으로 실행시켜 볼수 있다. 이렇게 한다면 오유가 나타나게 된 기본원인에 대해서 시각적으로 볼수 있을것이다. 오유가 나타난 코드를 삭제해 버리고 초기코드의 복사판이 아직도 남아 있는가를 확인해야 한다. 같은 행에서 오유가 최소로 되도록 입력할수 있다.

프로그람이 관계된다면 오유가 나타나는 원인을 이러한 행들에서 찾아 볼수 있다. 문제를 발생시키는 입력을 아는것은 쓸모 있는 정보이다.

오유수정: 드문드문 생기는 오유는 계속 추적하기가 힘들다. 만일 일치하게 일어 나지 않는다면 오유가 무조건 일어 나게 되다. 만일 오유추적이 어렵다면 프로그람을 계속 다시 실행해야 할것이다. 이러한 경우 매번 동작시켜야 한다.

프로그람에 따라 오유가 확고히 나타나게 하자면 어떻게 해야 하는가? 자원이 풍부해야 한다. 여기에 그 경우에 따라 프로그람작성자들이 사용하는 여러가지 일반기술이 있다.

오유를 발생시키기 위한 정확한 입력렬이 정확한 조건을 료해한다. 란수발생기를 리용한다면 매 실

행에서 결과가 같은 값이 되도록 한다.

프로그람이 오래동안 동작한 후에 오유가 생긴다면 동작을 모의하기 위한 방법을 료해한다. 아마 기억기루실이 있고 오유는 동적기억효률이 많을 때만 생기게 될것이다. 많은 기억기를 할당하는 기능을 써서 이 동작을 모의한다. 오래동안 동작하는 프로그람의 영향을 모의하기 위하여 프로그람의 앞에서 함수를 호출한다.

프로그람의 상태를 주기적으로 표시한다. 즉 열쇠값과 자료구조를 표시하시오. 프로그람을 초기화하여 수집한 이 마지막 《상태》정보를 사용하여 프로그람이 빨리 무조건 실패하도록 하시오.

오유지적: 문제를 발생시키는 함수나 코드부분을 결정한다. 이것을 위하여 자료값을 표시하고 불정확한 값이 생기는 실행부분을 찾는다. 또한 여러가지 입력값을 주고 코드에서 그 영향을 고찰한다. 흔히 정확한 입력에서 코드의 오유위치를 3각법으로 결정할수 있다. 그림 12-5에서 오유의 위치를 3각법으로 결정하는 과정을 보여 준다.

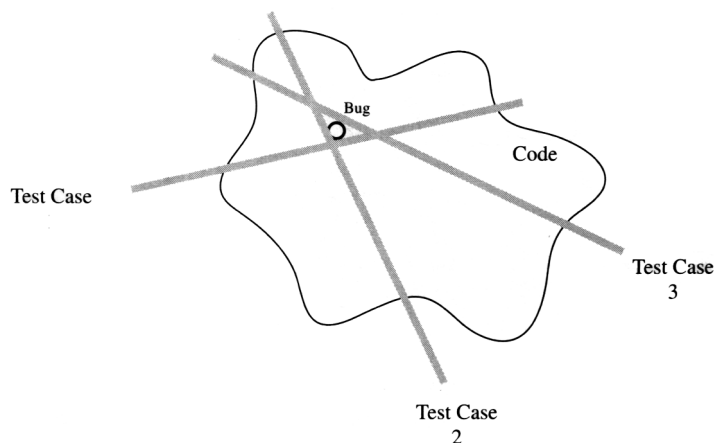



그림 12-5. 오유위치를 3각법으로 결정하기

오유의 위치결정 즉 3각법결정은 흔히 적은 도움이 요구되는 코드부분에 오유위치를 제한하는 작은 검사부분을 실행할것을 요구한다. 문제는 매 검사모임이 큰 부분코드를 판단한다는것이다. 따라서 2~3개 이상의 검사부분은 오유의 위치에 대하여 0을 요구한다.

일부 사람들에 대한 오유설명: HouseIcon실례의 일부 사람들에 대한 오유설명에서는 오유를 찾을 수 있다. 설명을 듣는 사람은 서로 다른 태도를 가지며 맹목적으로 문제를 본다.

만일 도움 받아 작업한다면 《자체오유수정》이라는 현상을 체험하게 된다. 한 사람이 문제를 설명하면 어떤 문제인지 선명해 진다.

사람들에게 코드를 설명하는것은 단계를 뛰여 넘지 못하며 더욱더 명백해 지게 된다. 자체오유수정은 놀랍게도 효과적이다 .

일반적인 오유인식: 여러가지 일반적오유는 특수한 증상을 가진다. 오유를 발생시키는 일반증상을 아는것은 그것들을 빨리 판단하는데 도움을 줄수 있다.

일반오유는 묶음신청초과이다. 이것이 나타나면 프로그람은 호출하려고 하지 않은 기억위치에 쓰거나 읽는다. 객체가 짝수값을 가지는 프로그람을 보는 임의의 순간에 객체가 원천코드를 본다. 탄창버리기는 또 하나의 일반적인 오유이다. 문제는 묶음의 신청초과나 부족에 의하여 생기게 되지만 국부일 때도 생기게 된다(실례를 들어 탄창에 할당된것). 증상은 이전 단락에 서술된바와 같이 서로 다르다. 탄창버리기증상의 하나는 함수귀환이 거짓을 발생시키거나 여러가지 이상한 위치로 함수가 귀환될 때이다(즉

실행은 방금 실행한 함수를 호출하지 않은 여러가지 기능으로 계속한다). 오유는 국부적인 묶음의 탄창위치가 덧쓰기되였기때문이다. 탄창에 기억된 한개 값은 현재함수가 돌려 줄 때 돌아 가는 함수의 주소이다.

오유수정기는 이 문제를 판단하는데 도움이 될수 있다. 수속은 함수가 귀환하기전에 정확한 정지점을 설정하는것이다. 귀환주소가 정확하다는것을 증명하기 위하여 루틴토막을 버린다. 만일 정확하지 않다면 그때 탄창은 이 함수에 의하여 호출되는 여러개 함수에 대하여 페기된다.

탄창버리기의 또 한가지 증상은 국부적인 객체가 이상하거나 불정확한 값을 얻는것이다. 여기서 탄창은 버렸지만 귀환주소는 없어 지지 않는다. 즉 함수호출의 국부를 버렸다.

고찰하는 함수에 대한 호출의 전과 다음에 정지점을 실정하는것은 탄창버리기이다. 첫 정지점에서 버려지는 국부객체값이 정확하다는것을 증명하시오. 실행을 계속하시오. 두번째 정지점에서 값을 재검사하시오. 틀리는가? 그렇다면 탄창버리기부분이 있다. 만일 이 값이 정확하다면 검토를 계속 하여야 한다.

빈 지적자에 대한 참조해제는 또 다른 하나의 대표적인 오유이다. 대체로 대부분은 빈 지적자가 참조될 때 제한이 생기게 되며 잘못된 원천명령문을 쉽게 판단할수 있다. 그러나 본질은 지적자가 왜 무효인가 하는것이다. 때때로 지적자는 무효로 되는데 사용자는 그때 if명령문으로 지적자에 대하여 참조해제를 진행하는 명령문을 보호해야 한다는것을 잊어 버린다.

모든것을 다시 콤파일하기: 현대적인 IDE(통합개발환경)는 아주 편리하지만 때때로 그것들은 혼동될수 있다. 만일 오유와 코드가 생각대로 되지 않는것을 보면서도 코드를 많이 바꾸었거나 변경이 크게 효과가 없다면 전체 대상과제를 재생하고 코드를 재실행하시오.

더 많은 정보수집: 무엇이 동작하는지 리해할수 없다면 더 많은 자료를 만드시오. 경험 있는 프로그람작성자들에 의하여 주어 진 여러가지 표준동작은 검사부분실행, 프로그람흐름을 수정하기 위한 오유수정기의 사용, 중간결과표시 그리고 자료구조버리기를 포함한다. 리해하는것만큼 선택하고 표시하여야 한다. 자료더미속에서의 검토는 풀더미속에서 바늘을 찾는것과 같다고 할수 있다.

콤파일러에 대한 주목: 현대적인 콤파일러는 오유의 실지형태를 발견하는데서 매우 좋다. 그러나 그것들은 심중히 지켜 보고 마지막에 허위인 경고통보를 만든다. 오유를 발견하면 무시되는 이러한 경고오유통보에 주목을 돌리는것이 좋다. 만일 객체가 설정되기전에 설정되였다는것을 콤파일러가 알린다면 그때 문제가 있는것을 검토한다. 가장 정확한 콤파일러설정은 오유검토에서 도움이 되는 정보를 내놓을수 있다.

오유수정:오유를 찾을 때 다른 오유가 흔히 나타난다. 이러한 오유는 찾으려는 오유에 무관계하게 보인다. 실지로 오유는 추적과정에 있으므로 발견된 오유에 대한것을 버리는것은 모험이다. 버린다면 추적이 힘들어 진다.

일반적으로 오유를 찾아 정리하는것은 좋은 습관이다. 찾은 오유는 실지로 찾으려고 하는 오유와 관계된다고 할수 있다. 이 오유를 수정하면 다른 오유들이 없도록 할수 있다. 이런 증상은 자주 나타난다. 다른 한편 새로 찾은 오유가 실지로 찾으려는 오유에 무관계한다면 그때 얻은것을 보관하는것은 응당하다.

중지하기:그것을 지적하기 위한 여러가지 방법을 리용하여 시간을 보냈음에도 불구하고 오유가 아무데도 없다. 인내성은 프로그람작성자의 중요한 특징이므로 중지할 때와 문제가 없을 때를 아는것도 또한 중요하다. 중요성을 아는 경험 있는 문제해결자는 다른데서 마음을 진정시킨다. 즉 흔들기, 음악듣기, 걷기, 창문내다보기 등이다.

대부분 경험 있는 프로그람작성자들은 오유찾기에 대한 재미나는 이야기를 가지고 있다. 일반적인 이야기는 《오유수정기의 출현》이다. 이야기는 다음과 같다. 빌 게크는 며칠동안 특이하며 불쾌한 오유

를 찾았다. 빌은 그만두고 자전거를 타면서 언덕들을 질주하기로 결심하였다. 나무들이 윙윙 소리가 나게 빌이 언덕을 날아 내려 가면서 그가 마지막으로 생각한것은 망열람기에서 피하기 쉬운 오유이다. 오후에 빌은 목욕하러 집으로 갔다. 목욕하는 동안 다시 오유에 대하여 생각하였는데 그것이 그의 머리를 쳤다. 즉 열람문제에 대한 해결은 명백해 졌다.빌은 자기의 가설을 실험하러 갔으며 그것을 인차 얻었다.

빌과 같은 이야기는 보통이다. 물론 가장 좋은 이야기들은 프로그람작성자들에게는 흔치 않다.

통의 외부고찰: 문제를 보거나 공격하는 특수한 방법으로 수정하기는 쉽다. 아무데도 문제가 없다면 통의 외부를 보시오. 전에 해보지 못한 드물거나 새로운것을 시험해 보시오. 실례를 들어 해오던것을 반대로 해보고 무엇이 생기는가를 보시오.

오유수정은 프로그람처리의 중요한 부분이다. 프로그람작성을 통하여 경험을 얻으므로 오유수정능력 즉 유능한 기술, 책략, 도구 등을 리용하는것은 반드시 필요하다. 또한 원천준위오유수정기의 사용방법을 배워야 한다. 현대적인 IDE로 구성된 원천준위오유수정기는 오유를 추적하는데서 위력한 수단이다. 그러나 기억해야 할 가장 중요한 점은 오유수정을 효과적으로 하기 위한 열쇠가 숙련된 처리에 관계된다는것이다. 이와 함께 숙련된 검사와 숙련된 오유수정은 고급쏘프트웨어가 시간과 투자에 관계된다는것을 명확히 한다.

문 제

15. 과학적인 방법의 단계들을 서술하시오.
16. 과학적인 방법은 론리적, 귀납적 혹은 추상적인 방법중에서 어느것을 쓰는가?
17. 변화되였을 때 중지되지 않도록 클라스 HouseIcon을 수정하시오.
18. 전역배렬에 대한 첨자화가 초과되는 프로그람의 동작을 보여 주는 프로그람을 작성하시오.
19. 탄창에 귀환주소를 넣는 프로그람의 동작을 보여 주는 프로그람을 작성하시오.
20. Therac-25사고를 검토하기 위하여 인터네트를 사용하시오. 무엇이 일어 났는가를 설명하시오.

12.3 알아 둘 점

- ✓ 검사는 쏘프트웨어에 오유가 전혀 없다는것을 증명할수 없다. 그것은 오유가 있다는것을 보여 주기만 한다.
- ✓ 오유는 4가지 종류로 구분된다. 즉 쏘프트웨어파괴나 자료변화, 정의에서의 만족이나 실패, 수행의 부족이나 불만족 그리고 사용의 어려움이다.
- ✓ 개발과정에 하는 검사가 빠르다. 초기에 찾은 오유를 수정하는것은 품이 덜 들고 더 쉽다.
- ✓ 프로그람을 완전히 검사한다는것은 불가능하다.
- ✓ 함수의 요구를 검사하기 위한 원형을 만드시오.
- ✓ 검토는 쏘프트웨어개발과정에 인차 오유를 찾아 내기 위한 효과적인 방법이다.
- ✓ 코드의 동작방법에 대한 지식이 없이 진행하는 검사를 검은 통검사라고 한다.
- ✓ 코드의 동작방법에 대한 지식을 가지고 하는 검사를 흰 통검사라고 한다.
- ✓ 동등구분에 대한 방조는 검사를 더 작고 더 효과적으로 되게 해준다.
- ✓ 좋은 검사모임은 쏘프트웨어의 경계조건을 검사하는 입력을 가진다.
- ✓ 명령문범위검사는 쏘프트웨어안에서 매 명령문이 한번만 실행하도록 하기 위하여 검사입력을 만든다.
- ✓ 경로범위검사는 쏘프트웨어에서 모든 조종흐름이 적어도 한번은 실행되게 하기 위하여 검사입력을

진행한다.

✓ 실행의 자동수속과 검사결과를 기억하여야 한다. 이것은 검사를 더 빠르게 해주며 개발자들과 애호가들을 위하여 처리과정을 문서로 제공한다.
✓ 오유수정에서 과학적인 방법을 리용해야 한다.
✓ 오유수정시간을 충분히 잡아야 한다. 오유수정을 빨리 하기 위하여 코드를 실행해 보면서 하는것이 좋다. 쏘프트웨어개발에서 오유수정시간은 앞으로 예견될수 있는것까지 고려하여 투자하여야 한다.
✓ 오유수정시 프로그람에 간단한 값을 입력하여 오유를 찾아 낼수 있다.
✓ 항상 반복되는 오유가 나타나는가를 확인해야 한다. 이것은 오유를 쉽게 리해하고 오유처리를 빨리 할수 있게 한다.
✓ 일반적인 오유의 형태들을 알고 있어야 한다.

참고할 책

검사와 오유수정에 대하여 쓴 유명한 책들이 많다. 검사에 취미가 있는 학생들은 다음의 책들을 참고할수 있다.

- Brian Kernighan and Rob Pike , The Practice of Programming, Reading, MA: Addison-Wesley, 1999.
- Steve Mc Connell, Code Complete: A Practical Handbook of Software Construction, Redmond, WA: Microsoft Press, 1993.
- Glenford Myers, The Art of Software Testing, New York: Wiley, 1979.
- Ron Patton, Software Testing, Indianapolis: Sams Publishing, 2001.

련습문제

12.1 Ariane 5 의 파괴는 숙련이 부족한 결과 생겼다. 서고를 리용하거나 인터네트를 리용하여 Ariane5의 파괴에 대하여 알아 보고 이 검사처리를 어떻게 하였는가를 설명하시오.

12.2 프로그람 5-1을 검사하기 위한 검사입력값을 만드시오. 검사입력값은 프로그람의 전체 명령적용범위를 만든다. 즉 검사입력값은 실행될 프로그람에서 적어도 한번은 있다.

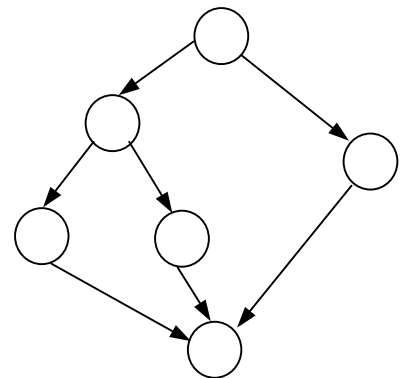
12.3 검토자료는 오유조건을 검사하기 위한 입력값으로 구분된다. 또한 유효한 자료가 주어 졌을 때 프로그람이 동작하는 시험값을 입력한다. 프로그람 5-3을 검토하시오. 오유조건의 검사입력값모임을 만들고 프로그람작업을 보여 주는 입력값모임을 분리하시오.

12.4 다음과 같은 조종흐름도에서 유일한 경로들은 몇개나 있는가?

12.5 검사장치를 개발하고 목록 9-18의 Display클라스에 대한 입력값을 검사하시오.

12.6 4명의 동무들과 함께 8장에서 보여 준 붉은색-노란색-풀색유희의 검토를 수행하시오. 검토에서 발견된 임의의 문제를 서술하는 검토보고서를 준비하시오.

12.7 프로그람을 검사하는데서 프로그람작성자가 검열하는것보다 다른 사람이 검열하는것이 왜 더 좋은가를 설명하시오.

12.8 다른 검사방법도 적용하시오. 어떤 방법을 적용할수 있는가?

제 13 장. 계 승

소개

객체지향언어의 기본특징은 계승이다. 계승은 이미 존재하는 클라스를 기초로 리용하여 새로운 클라스를 정의하는 기능이다. 새로운 클라스는 자기가 기초하는 클라스의 속성과 동작을 계승하며 또한 자기에게 특정한 속성과 동작을 창조할수 있다. 계승은 믿음성 있고 리해하기 쉬우며 원가가 눅고 적응성이 높으며 재리용할수 있는 프로그람을 제작하기 위한 기본틀을 제공하는 강력한 기구이다. 이 장에서는 두개의 지수를 표시하고 현시하기 위한 클라스묶음을 개발하는것으로써 C++계승기능을 소개한다. 결과클라스들은 7장에서 개발하는 만화경프로그람을 개선하고 확장하는데 리용될수 있다.

개 념

- is-a관계
- has-a관계
- 사용관계
- 기초클라스
- 파생클라스
- 공개부계승
- 비공개부계승
- 단일계승
- 다중계승

13.1 계승을 리용한 객체지향설계

생물학에서의 계승의 개념에 대해서는 모두다 잘 알고 있다. 생물은 다 자기 조상으로부터 특징을 계승한다. 실례로 우리는 그것을 바라든 바라지 않든 부모들에게서 여러가지 특징들을 계승받는다. 이 특징들은 자기 부모들에게 고유한 특징일수도 있으며 혹은 자기 조부모들의 특징들을 계승한것일수도 있다. 계승의 개념은 복잡한 체계를 설계하는데 리용될수 있다. 계승은 체계를 리해하는데 도움이 되는 기층적인 구조로 체계를 구성하는 한가지 방법을 제공한다. 또한 그것은 코드를 재리용하기 위한 틀을 제공한다.

실제적으로 계승은 가장 일반적인것부터 드문것에 이르기까지 우에서 아래로 내려 가면서 서술하는 방식으로 추상를 정립한다. 그림 13-1은 여러가지 문서편집도구의 계층구조를 보여 준다.

실례로 원주필, 볼원주필, 소묘연필 그리고 만년필과 같은 서로 다른 펜의 형태들을 볼수 있다. 이것은 모든 펜들의 특수한 형태들이며 그것들은 펜의 특징들을 계승한다. 즉 잉크를 쓴다. 어쨌든 펜의 매 형태들은 다른 형태로부터 펜을 식별하고 펜에 대한 개별적인 속성을 가진다.

펜의 제일 높은 준위에서 구별되는 특징은 종이우에 잉크로 쓰는 기구라는것이다. 확실히 추상에 대한 계층도를 창조하는데 계층표기를 리용하는것은 추상들사이의 관계를 리해하는데 도움을 주지만 또 다른 하나는 새롭고 쓸모 있는 추상을 만드는 노력을 줄이는데 도움을 준다는것이다.

계승에서 기초하고 있는 추상의 계층도를 구성하는데 도움을 주는 관계는 is-a관계이다. is-a관계는 하나의 추상이 다른것에 대한 특수화라는것을 의미한다. 실례로 펜은 특수한 필기도구이다. 또한 볼원주

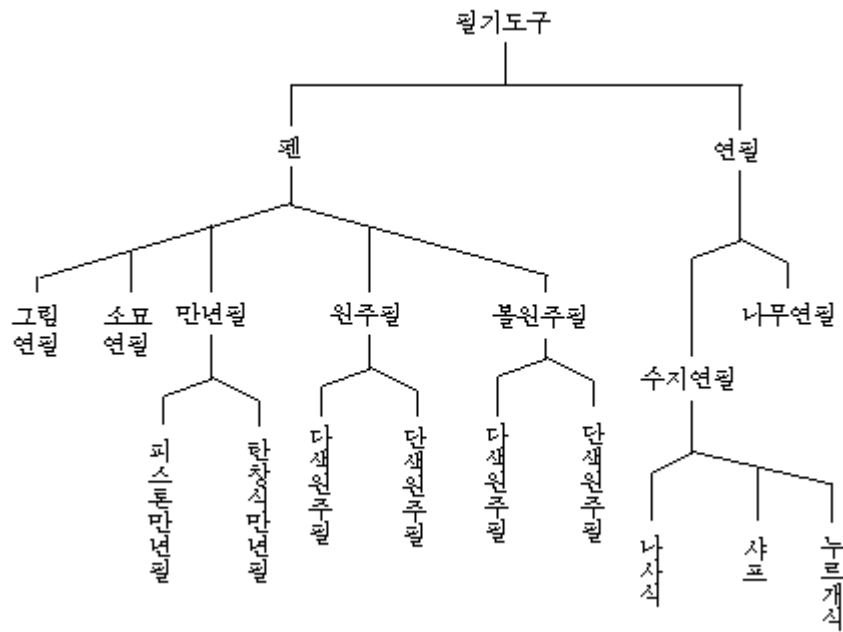



그림 13-1. 필기도구의 구분

필형태도 필기도구의 한 형태이다.

is-a관계는 중간상태이지만 그것은 반영하지 않는다는것을 참고하여야 한다. is-a관계를 표현하는 다른 방법은 볼원주필은 펜의 한 종류이며 펜은 필기도구의 한 종류라는것을 보여 준다. is-a관계는 추상사이관계와 추상사이계승을 표시한다. 그 하나는 우리가 이미 리용하여 온 has-a관계라고 한다. 이 관계에서는 여러개의 객체를 부분적으로 본다. 실례로 만년필은 펜촉을 가지고 일부 연필들은 지우개를 가진다.

has-a관계에서는 포함이 절대적이다. 자동차는 기관, 충전지, 경적을 포함한다. 9장에서는 Location추상을 포함한 여러가지 추상들을 소개하였다. 이 추상들은 has-a관계를 리용하였다. Maze는 시작위치와 끝위치를 가진다. 따라서 Maze는 Location을 가진다. 이것과 류사하게 Wanderer는 위치를 가진다. 따라서 Wanderer는 Location을 가진다.

객체들사이의 다른 관계는 uses-a관계이다. 이 관계는 하나의 객체가 다른 객체를 여러가지 방법으로 리용한다는것을 보여 준다. 객체지향프로그람에서 이 관계는 성원함수를 거쳐서 다른것으로 전달하는 하나의 객체에 의하여 실제적으로 실현된다. 실례로 쓰고 있는 조작체계가 현재의 시간과 날자를 유지하는 시계객체를 가진다고 가정하시오. 시계객체는 현재의 날자와 시간을 돌려 주는 성원함수를 가진다. 날자 및 시간을 요구하는 다른 객체들은 현재의 시간이나 날자를 얻기 위하여 해당성원함수를 호출하는 것으로써 시계객체를 리용한다.

13.2 계승의 재리용

7장에서는 만화경으로 제작된 그림을 모의하는 프로그람을 개발하기 위하여 클라스 RectangleShape를 리용하였다. 프로그람에 의하여 만들어 진 패턴들은 훌륭했지만 대부분의 만화경들은 다색패턴들을 만들기 위하여 한가지 유리형태보다 더 많이 리용되였다. 원, 삼각형과 같은 견본들을 더 리용하기 위하여 만화경프로그람을 발전시켜야 한다.

한가지 방법은 원을 현시하기 위한 클라스를 추가적으로 만드는것이다. 견본에서와 같이 클라스 RectangleShape를 리용한다면 클라스 CircleShape를 구축하는것은 그리 어렵지 않다.

목록 13-1은 본래의 RectangleShape클라스에 대한 선언과 새로운 CircleShape클라스에 대한 선언을 보여 준다.

목록 13-1. CircleShape클라스와 RectangleShape클라스의 선언

```
class RectangleShape {
    public:
        RectangleShape(SimpleWindow &Window,
        float Xcoord, float Ycoord, const color &Color,
        float width, float Height;
        void Draw();
        color GetColor() const;
        void GetSize(float &Width,  float &Height)  const,
        void GetPosition(float &Xcoord, float &Ycoord)  const;
        float GetWidth()  const;
        float GetHeight()  const;
        SimpleWindow& GetWindow()  const;
        void SetColor(const color  &color);
        void SetPosition(float Xcoord, float Ycoord);
        void SetSize(float Width, float Height);
    private:
        SimpleWindow &Window;
        float Xcenter;
        float Ycenter;
        color Color;
        float Width;
        float Height;
};
class CircleShape {
    public:
        CircleShape(SimpleWindow &Window,
        float Xcoord, float Ycoord,
        const color &Color, float Diameter);
        void Draw();
        color GetColor() const;
        float GetSize(0 const;
        void GetPosition(float &Xcoord, float &Ycoord)  const;
        SimpleWindow& GetWindow()  const;
```

```
        void SetColor(const color &Color);
        void SetPosition(float Xcoord,  float Ycoord);
        void SetSize(float Dameter);
    private:
        SimpleWindow &Window;
        float Xcenter;
        float Ycenter;
        color Color;
        float Diameter;
};
```

두 정의가 매우 류사하다는데 주의를 돌리시오. 두 클라스들은 꼭 같은 성원함수들을 가진다. 성원함수 SetSize()만이 두 클라스사이에 약간 차이난다.

RectangleShape클라스는 SetSize()에 대한 두 float형파라메터 즉 Width와 Height를 가지며 그리고 CircleShape는 Diameter라는 1개의 float형파라메터를 가진다. 이와 류사하게 두 클라스들의 매개자료성원은 크기속성을 얻기 위하여 CircleShape가 Diameter를 리용할 때마다 RectangleShape가 Width와 Height에 크기속성을 기억하는것을 제외하고는 같다.

물론 CircleShape의 성원함수들의 실행을 제공하는것도 필요하다. 다시 말하여 이것은 대부분의 코드가 RectangleShape에 대하여 개발된것과 같기때문에 그리 어렵지 않다. 성원함수 Draw()만이 크게 차이난다. 따라서 두 클라스사이에는 많은 공통점이 있다.

현재까지는 매우 좋았지만 지금은 다른 표본들을 가지고 있어야 한다고 가정해 보시오. 실례로 4각형과 원뿐아니라 3각형을 리용하는것도 좋다. 작업은 복잡하고 지저분하다. 그와 류사하거나 같지 않은것들도 많다. 더우기 표본에 대한 새로운 조작이나 속성을 추가하려면 모든 표본을 변경해야 하므로 더 어렵다. 계승을 리용하여 표본의 추상을 설계한다면 설계와 실행에서 복잡성을 줄일수 있다. 계승을 리용하면 올림법보다는 오히려 내림법으로 설계를 해나가는것이 좋다. 즉 일반적인것을 리용하여 표본에 대한 대부분의 일반추상을 생각해야 하며 더 많은 특수한 추상을 만들어야 한다.

대부분의 일반추상이나 기본추상을 결정하는것은 추상의 계승에서 객체지향설계를 성과적으로 할수 있게 하는 한가지 열쇠이다. 추상의 유연한 체계를 설계하는것은 매우 어렵다. 그것은 너무 쉬워 기본추상을 특별히 또는 많이 만들수 없다. 기본추상이 아주 특수하다면 is-a관계는 체계에서 어미의 속성을 계승하기때문에 기본추상으로부터 만들어 지거나 파생되여야 하는 추상들사이에는 유지될수 없다. 그렇기때문에 실례로 려객운반수단형태의 계층에 대한 기본추상이 바퀴수에 대한 속성을 가진다면 계승에서 아래의 모든 추상들은 이 속성을 계승한다. 따라서 이 추상은 발판을 리용하는데서와 같이 바퀴가 없이 운반수단에 대한 추상을 파생하는데 리용할수 없다.

만일 기본추상이 명확하지 못하면 속성과 조작이 파생된 추상에서 필요없이 복사된다. 실례로 만일 려객운반수단의 계층에 대한 기본추상이 손님수에 대한 속성을 빼놓는다면 이 속성은 그 기본추상으로부터 파생되는 운반수단의 매 종류에 추가되여야 한다.

13.3 도형의 계층

계승에 기초한 추상의 계층발전을 설명하기 위해서 2차원도형들과 본문표식들로 이루어 진 창문객체들의 모임을 위한 클라스계층을 만든다. 두가지 작업으로 확장된 만화경프로그람을 설계할수 있다. 첫 작업은 기초클라스를 포함한 속성과 조작들이 무엇인가를 결정하는것이다. 왜냐하면 그 방법이 도형체계작업을 창문화하는 경우 본문이나 창문은 그것을 현시할것을 알려 주는 하나의 객체를 포함하기때문이다. 그러므로 하나의 창문객체의 속성은 그것을 포함하는 창문이다. 매개 창문객체에서 다른 속성은 창문의 위치이다.

크기, 색갈과 같은 다른 속성들이나 어떤 객체가 그려 지는가 하는것은 객체의 형에 의존하므로 기초클라스의 부분이 아니다. 물론 그 속성들외에 속성들을 결정하고 다루는 공개성원함수들이 필요하다. 그림 13-2에서는 창문객체추상을 보여 주었으며 목록 13-2는 그에 대한 클라스선언을 보여 준다.

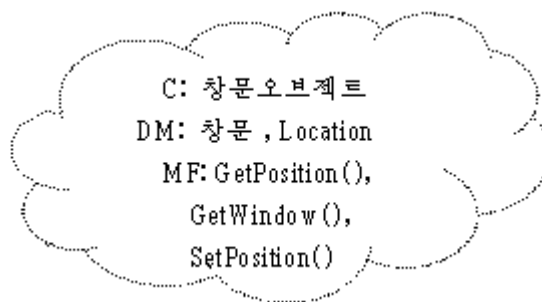



그림 13-2. WindowObject추상화

목록 13-2. WObject.h에서 WindowObject클라스선언

```
#ifndef WINDOWOBJECT_H
#define WINDOWOBJECT_H
#include "ezwin.h"
//창문에 현시될수 있는 객체를 위한 WindowObject기초클라스
class WindowObject {
    public:
        WindowObject(SimpleWindow &w, const Position &p);
        Position GetPosition()  const;
        SimpleWindow&  GetWindow() const;
        void SetPosition(const Position &p);
    private:
        SimpleWindow &Window;
        Position Location;
};
#endif
```

WindowObject클라스는 Position과 SimpleWindow클라스들의 구체레인 Location과 Window라는 2개의 비공개성원항목들을 포함하는데 그것은 10장에 소개되였다. 그 자료성원 Window는 하나의 참조

객체이다. 참조객체들은 변경될수 없다. 따라서 일단 Window가 특별히 SimpleWindows로 설정되면 그것은 항상 그 SimpleWindow로 참조된다. WindowObject가 하나의 창문으로 제한되면 다른 Window에 현시되거나 움직이지 말아야 한다.

WindowObject의 실현부는 간단하고 정확하며 목록 13-3에 주어 졌다. 구축자는 본체가 없으며 성원함수의 실현부는 한행짜리 함수라는것을 명심하시오. 이 기초적인 추상 WindowObject로부터 2차원평면과 본문표제들에 대한 추상을 파생할수 있다. 먼저 2차원평면을 보자. 모든 2차원평면들이 가지는 추가적인 속성은 색이다. 그래서 기초추상 WindowObject로부터 색을 가진 Shape라는 새로운 추상을 파생한다.

Shape는 WindowObject로부터 파생되기때문에 WindowObject자료성원(례하면 Window와 Location)과 해당성원함수(례하면 Get Position(), GetWindow()와 SetPosition())를 계승한다. 기초클라스와 파생클라스사이관계에 대해서 고찰하는 다른 방법은 기초클라스가 그것으로부터 파생된 모든 클라스들에 대한 공동봉사모임을 제공하는것이다.

목록 13-3. WObject.cpp에서 WindowObject의 실현부

```
#include "wObject.h"
WindowObject::WindowObject(SimpleWindow &w,
  const Position &p) : Window(w),   Location(p) {
    //코드가 필요없다.
}
Position WindowObject::GetPosition() const {
    return Location;
}
SimpleWindow&  WindowObject::GetWindow() const {
    return Window;
}
void WindowObject::SetPosition(const Position &p) {
    Location = p;
}
```

WindowObject는 창문에 현시되는 모든 객체들에 필요한 기초적인 봉사를 제공한다.

Shape가 WindowObject의 동작과 속성들을 계승하는 하나의 계승을 가지게 된다. 이미 계승에 대하여 론의하였지만 계승은 다중화될수 있다. 다른 계승을 리용하여 클라스 Shape로부터 특수한 표본형태들을 파생할수 있다. 쓸수 있는 3가지 기초표본들은 4각형, 타원, 삼각형들이다. 특정한 매 표본은 그 크기가 약간씩 차이나므로 표본의 크기를 기억하기 위한 자체의 자료성원들을 가진다. 그림 13-3은 매 표본형태에 대하여 요구된 크기에 따르는 자료성원들을 보여 준다. 즉 파생된 표본은 화면상에서 표본을 묘사하고 그리는 방법이 어떤 표본형태인가 하는데 의존하므로 자체그리기성원함수를 가지게 된다.

따라서 EllipseShape는 특정한 두개의 자료성원들을 가진다. Width와 Height는 타원의 가로와 세로직경을 나타낸다. RectangleShape도 2개의 자료성원 Width와 Height를 가지는데 그것은 4각형의 너비와 높이이다. TriangleShape는 간단히 하기 위하여 모든 면들이 같은 길이를 가진 바른3각형이다.

따라서 TriangleShape는 하나의 자료성원 SideLength만을 가진다.

그림 13-4는 창문객체들의 계층구조를 보여 준다.

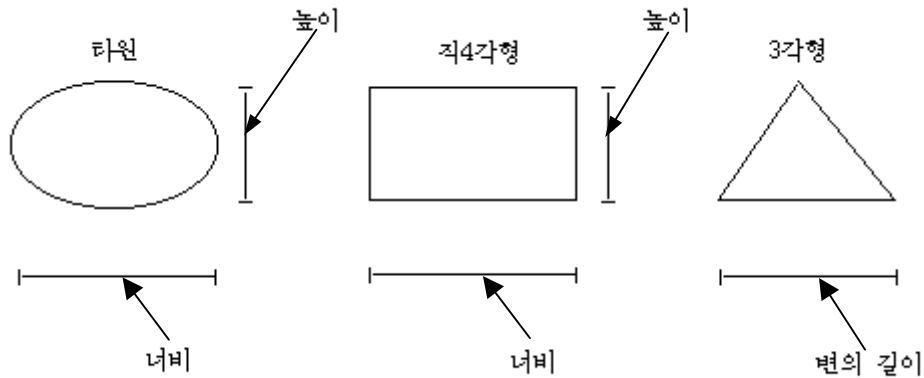
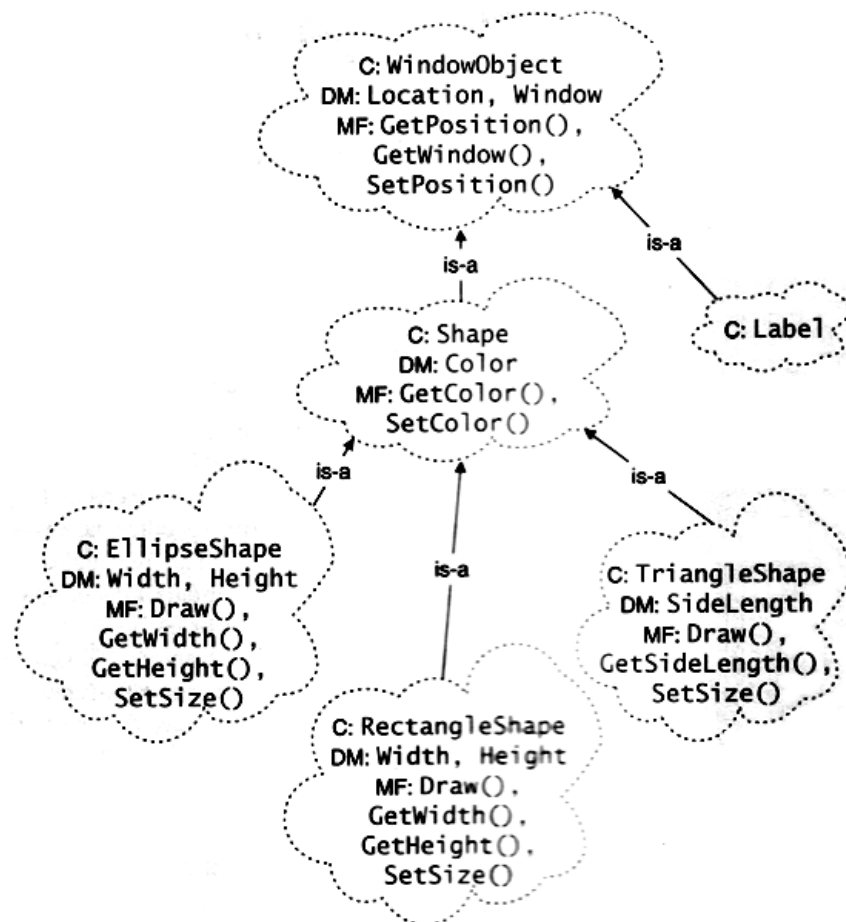



그림 13-3. 도형과 그것들의 크기자료성원

그림 13-4. 창문객체의 계층구조

경험

내부전개성원함수

대체로 클라스를 2개의 구성요소로 나눈다. 즉 클라스에 대한 대면부(.h파일)와 클라스의 실현부(.cpp파일)이다. WindowObject에 대한 대면부는 WObject.h에 포함되고 실현부는 WObject.cpp에 포함된다.

일반적으로 클라스를 대면부(.h파일)와 실현부(.cpp파일)로 분할하는것은 좋은 방법이다. 그러나 때때로 이렇게 하지 말아야 하는 경우도 제기된다. 흔히 능률을 높이기 위하여 대면부에 실

현부를 포함시키게 된다. 실례로 아래의 코드에서는 .h파일안에 WindowObject의 실현부가 포함되여 있다.

```
class WindowObject{
    public:
        WindowObject(SimpleWindow &w,
        const Position &p);
        Position GetPosition()  const;
        SlmpleWindow   &GetWindow()    const;
        void  SetPosition( const  Position &p);
    private;
        SimpleWindow &Window;
        Position Location;
};
Inline WindowObject::WindowObject (SimpleWindow &w,
  const Position &p); Window(w), Location(p){
//코드는 필요되지 않는다.
}
Inline Position WindowObject::Getposition() const{
    return Location;
}
Inline SimpleWindow& WindowObject::GetWindow()
  const {
return Window;
}
Inline void WindowObject:: SetPosition(const Position
  &p) {
    Location=p;
}
```

예약어 Inline이 구축자와 성원함수의 정의에 추가되였다. Inline변경자는 콤파일러에 함수에 대한 호출을 적당히 치환된 파라메터들을 가진 함수의 실제적인 본체로 바꾸도록 지시한다. 이 방법을 쓰면 함수호출에 대한 그리고 많이 리용된 성원함수들에 대한 간접조작시간에 의하여 프로그람의 속도가 훨씬 더 빨라 지는것을 막을수 있다. 내부전개(Inline)는 클라스정의에서 성원함수를 정의하는 방법으로도 실현할수 있다. 이 방법은 대면부와 실현부를 더욱더 굳건히 결합시킨다. 이에 대해서는 설명하지 않는다.

13.3.1 파생클라스의 선언

파생클라스를 선언하자면 기초클라스를 렬거하기 위하여 클라스선언문법에 대한 간단한 추가가 필요하다. 파생클라스를 선언하는 문법은 다음과 같다.

```
파생클라스이름      접근지정자          기초클라스이름

class Derivedclass: public Baceclass{
    public;
        //공개부분
        ...
    private;
        //비공개부분
        ...
};
```

다 아는바와 같이 접근지정자(access specifier)는 보통 기초클라스의 공개부성원들이 파생클라스의 공개부성원으로 된다는것을 의미하는 **public**이다. 이 관계를 공개부계승(public inheritance)이라고 한다. 또한 비공개와 보호부계승도 있다. 이 관계들은 드물게 리용된다. 아래의 코드는 클라스 Shape를 보여 주는데 그것은 기초클라스 WindowObject로부터 파생된다.

```
class shape: public WindowObject{
    public :
        Shape ( SimpleWindow &w, const Position &p,
        const color &c=Red);
        color GetColor() const;
        void SetColor( const color &c);
    private:
        color Color;
};
```

C++프로그람작성자는 흔히 이것을 《Shape는 창문객체의 한 종류이다.》라고 한다(is-a관계).

첫 행을 변경시키는것과는 달리 Shape의 선언은 하나의 새로운 클라스를 선언하기 위한 표준기술에 따라 간다. 그것은 공개성원함수들과 함께 구축자를 포함한다. 표본을 위한 구축자는 2개의 요구파라메터 즉 표본을 포함한 창문과 표본을 그리기 위한 위치를 가진다. 색에 대한 파라메터는 기정값으로 제공된다.

그 구축자외에 Shape는 두개의 공개성원함수들을 가진다. 검토자 GetColor()는 표본의 색을 받고 변이자 SetColor()는 색을 변경한다. 마지막으로 Shape는 하나의 비공개자료성원 Color를 가진다. 이 자료성원은 표본의 색을 가진다.

클라스표본으로부터 구별되는 표본들에 대한 클라스들을 만들수 있다. 아래의 코드는 RectangleShape를 선언하는데 Shape로부터 파생된것이다.

```
class Rectangle Shape : public Shape{
    public:
        Rectangle Shape (SimpleWindow &Window,
        const Position &Center, const color &c=Red,
        float Width=1.0, float Height=2.0);
        float GetWidth(0 const;
        float GetHeight(0 const;
        void Draw();
        void SetSize (float Width, float Height);
    private:
        float Width;
        float Height;
};
```

그의 구축자외에 RectangleShape는 4개의 공개성원함수를 가진다. 2개의 검토자들은 4각형의 너비와 높이를 받으며 SetSize()는 너비와 높이를 설정하기 위한 변이자이다. 마지막으로 성원함수 Draw()

는 4각형을 그린다. 이 성원함수들은 그것들의 동작(무엇을 하는가)이 표본의 개별적인 형에 관계되므로 RectangleShape로 렬거한다. 실례로 4각형을 그리는 방법은 타원이나 3각형을 그리는 방법과 다르다. 같은 방법으로 표본의 크기를 렬거하는데 필요한 파라메터들도 표본의 형태에 의존한다.

마지막으로 RectangleShape는 2개의 비공개자료성원 Width와 Height를 가진다. RectangleShape 초기판과 같이 성원들은 4각형의 너비와 높이로 된다.

RectangleShape와 함께 또한 EllipsShape와 TriangleShape추상들을 만들어야 한다. 그것들의 선언은 RectangleShape의 선언과 꼭 같다. 목록 13-4는 파생클라스 EllipseShape에 대한 선언을 보여 준다.

목록 13-4. Ellipse.h에서 EllipShape의 선언

```
#ifndef ELLIPSESHAPE_H
#define ELLIPSESHAPE_H
#include "Shape.h"
class EllipseShape : public Shape {
    public:
        EllipseShape(SimpleWindow &Window,
        const Position &Center, const color &c = Red,
        float Length = 1.0,  float Height = 2.0)
        float GetWidth() const;
        float GetHeight()  const;
        void Draw();
        void SetSize(float Width, float Height);
    private:
        float Width;
        float Height;
};
#endif
```

EllipseShape의 선언이 RectangleShape와 거의 같다는것을 알수 있다. 차이점은 클라스의 이름과 비공개성원자료뿐이다. 물론 성원함수 Draw()의 실현부는 다르다.

목록 13-5는 클라스 TriangleShape의 선언을 포함한다. 다시 말하여 선언은 다른 표본들과 류사하다.

공개성원부분은 구축자, 그리기함수, 크기설정함수를 포함한다. 그러나 3각형을 위하여 비공개부분은 하나의 자료성원 즉 3각형의 변들의 길이를 포함한다. 바른3각형으로 작업하기로 하였으므로 하나의 변만이 필요된다.

목록 13-5. Triangle.h에서 TriangleShape의 선언

```
#ifndef TRIANGLESHAPE_H
#define TRIANGLESHAPE_H
#include "Shape.h"
```

```
class TriangleShape : public Shape {
    public:
        TriangleShape (SimpleWindow &w, const Position &p,
        const color &c=Red, float SideLength = 1.0);
        float GetSideLength() const;
        void SetSize(float SideLength);
        void Draw();
    private:
        float SideLength;
};
#endif
```

13.3.2 파생클라스의 실현부

앞에서 본바와 같이 파생클라스의 선언은 기초클라스의 선언과 거의 류사하다. 마찬가지로 파생클라스의 실현부는 기초클라스의 실현부와 그리 차이가 없다. 명심해야 할것은 파생클라스구축자가 본질적으로 기초클라스객체를 파생클라스객체로 만드는데 필요되는 기능을 바로 그 기초클라스객체에 추가한다는것이다. 즉 파생클라스는 기초클라스의 전문화이다. 따라서 기초클라스를 위한 구축자는 파생클라스를 위한 구축자가 동작을 할수 있기전에 기초클라스객체를 창조할수 있도록 호출되여야 한다. 계승에 대하여 생각할 때 그 순서가 문제로 제기된다. 기초클라스객체는 그것이 파생클라스객체로 변환되기전에 존재해야 한다.

파생클라스의 구체례가 창조되였을 때 기초클라스의 구체례를 창조하는데 편리한 기능을 제공하기 위하여 구축자가 렬거하는 방법을 약간 확장하여야 한다. 파생클라스의 구축자에 대한 문법은 다음과 같다.

첫번째 행은 기초클라스구축자에 대한 호출을 의미한다. 실례로 Shape구축자의 실현부는 다음과 같다.

```
Shape::Shape (SimpleWindow &w, const Position &p,
  const  color &C):  WindowObject ( w. p), Color(C) {
    //코드는 필요없음!
}
```

이 구축자는 Shape객체가 정의될 때 WindowObject구축자를 호출하며 창문과 위치파라메터 W, P를 받아 그러한 속성을 가진 WindowObject를 만들수 있다는것을 지적한다. 기초클라스를 위한 구축자

는 파생클라스의 구축자가 실행되기전에 호출된다. 기본객체가 파생객체의 기초를 형성하므로 이 순서가 문제로 된다(파생객체로 그것을 바꾸기전에 기본객체를 요구한다). 기본객체가 구체레화되고 파생클라스로 변화된 다음 자료성원초기화목록의 임의의 자료성원구축자를 불러 낸다. Shape의 속성 Color는 이런 방법으로 초기화된다. 마지막단계에서 구축자의 실제코드를 실행한다. Shape의 경우에 기초클라스의 구축자와 자료속성 Color구축자를 불러 낸다. 이에 의해서 모든 동작이 조종된다. 그러므로 코드가 필요없다.

경험

자료성원초기화목록의 사용

클라스구축자가 실행될 때 자료성원을 초기화하는 방법에는 두가지가 있다. 한가지 방법은 구축자안에서 자료성원의 변이자를 호출하는것이다. 다른 방법은 자료성원초기화목록에서 자료성원의 구축자를 호출하는것이다. 대체로 두번째 방법이 많이 리용된다. 첫번째 방법에서 자료성원기정구축자는 객체를 기정값으로 만들기 위하여 불러 낸다. 그리고 기정값을 초기값으로 다시 설정하기 위하여 변이자가 호출된다. 자료성원초기화목록방법을 리용하여 자료성원은 해당한 초기값으로 된다.

Shape로부터 파생된 클라스의 구축자도 이와 같다. 그러나 이때 지정한 표본구축자는 Shape구축자와 WindowObject구축자를 리용하는것으로 된다. 실례로 RectangleShape의 구축자는 다음과 같이 수행된다.

```
RectangleShape::RectangleShape(SimpleWindow &Window,
  const Position &Center, const color &c, float w,
  float h):Shape(Window, Center, c ),
Width(w), Height(h){
  //코드가 필요없음.

}
```

이 구축자는 RectangleShape객체가 정의될 때 Shape구축자가 호출되는데 창문, 위치, 색파라메터가 쓰인다는것을 정의한다. 물론 Shape구축자는 창문을 그리고 위치자료성원을 초기화하기 위하여 WindowObject를 리용한다. 실례로 정의

```
RectangleShape Lawn(TWindow, LeftCorner, Green, 2.5, 3.5);
```

가 실행될 때 콤파일러에 의하여 진행되는 처음동작은 변수 TWindow, LeftCorner, Green을 가진 Shape구축자를 불러 내는것으로부터 시작된다. Shape함수구축자는 TWindow와 LeftCorner값을 가진 WindowObject의 구축자를 리용한다. 이때 WindowObject가 구체레화되며 Window와 Location구축자를 불러 내며 WindowObject구축자의 빈 구체레를 불러 낸다. 그때 Shape객체는 구체레화되고 Color구축자가 값 c로 초기화되기 위하여 호출된다. 다음 Shape구축자구체레가 실행되면 이 구축자는 코드를 요구하지 않는다. 이와 같은 단계를 거쳐 RectangleShape Lawn이 구체레화되며 Width와 Height구축자가 초기화되기 위하여 호출된다. 마지막으로 RectangleShape구축자의 구체레를 불러 낸다. 어미클라스구축자와 자료성원의 구축자가 모든 작업을 하기때문에 코드가 필요없다.

다른 파생클라스의 구축자들도 같다. 타원을 위한 구축자는 아래와 같다.

```
EllipseShape::EllipseShape(SimpleWindow &Window,
  const Position &Center, const color &c, float w,
  float h) : Shape (Window, Center, c),
```

```
Width (w), Height(h){
    // 코드가 필요없음!
}
```

3각형의 클라스도 이와 같다.

```
TriangleShape::TriangleShape (SimpleWindow &Window,
  const Position &p, const color &c, float l)
  : Shape(Window, p,c), SideLength(l) {
    //코드가 필요없음!
}
```

이 구축자들에 대한 참조는 아주 간단하다. 사실상 구축자의 구체례를 위한 코드작성은 필요없다. 그 리유는 어미클라스 Shape의 개발코드가 리용될수 있기때문이다. 비록 클라스들이 단순하고 보호가 잘 되지 않았다고 하더라도 앞으로 더 복잡하고 보호가 잘 된 클라스에 대하여 어지간한 표상을 가질수 있다.

파생클라스의 성원함수실현부를 정의하는 문법은 기초클라스의 성원함수실현부를 정의하는 문법과 같다. 그 문법은 다음과 같다.

```
Type Cname :: Mfunction(Plist){
    //파생클라스의 성원함수의 구체례
    ...
};
```

실례로 RectangleShape의 SetSize()성원함수는 다음과 같이 동작한다.

```
void RectangleShape::SetSize(float w, float h){
    Width = w;
    Height = h;
    return ;
}
```

이 함수는 RectangleShape의 크기속성값 Width와 Height를 변화시킨다. RectanglesSape의 Draw()성원함수도 이와 류사하게 동작한다.

```
void RectangleShape::Draw() {
    const Position Center = GetPosition();
    const float Width = GetWidth();
    const float Height = GetHeight();
    const Position UpperLeft = Center
      +Position (-.5*Width, -.5*Height);
    const Position LowerRight  = Center
      +Position(.5*Width, .5*Height);
```

```
        GetWindow(), RenderRectangle(UpperLeft, LowerRight,
        GetColor());
        return;
    }
```

RectanlgeShape의 그리기성원함수는 RenderRectangle()을 호출하는데 그것은 EzWindows클라스 SimpleWindow의 성원함수이다. RenderRectangle은 창문에 4각형을 그린다. 이 함수의 파라메터는 왼쪽웃구석과 오른쪽아래구석에 대한 자리표값과 색값이다. SimpleWindow의 아래준위성원함수들은 RenderEllipse()와 RenderPolygon()이다. RenderEllipse()는 RenderRectangle()과 같다. 파라메터는 타원외접4각형의 자리표와 타원의 색이다. 이 외접4각형은 타원을 포함하는 4각형이다. 이 관계를 그림 3-5에 보여 준다.

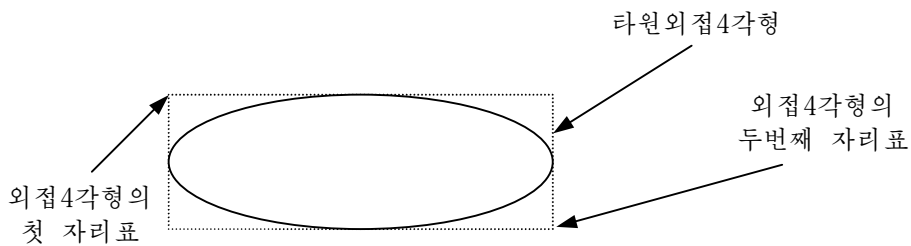



그림 13-5. 타원을 포함하는 4각형과 타원의 관계

EllipseShape의 Draw()함수는 RectangleShape의 Draw()와 동작이 비슷하다. 코드는 다음과 같다.

```
    void EllipseShape :: Draw(){
        const Position Center = GetPosition();
        const float Width = GetWidth();
        const float Height = GetHeight();
        const Position UpperLeft = Center
          +Position(-.5*Width, -.5*Height);
        const Position LowerRight = Center
          +Position(.5*Width, .5*Height);
        GetWindow().RenderEllipse ( UpperLeft, LowerRight,
          GetColor());
        return;
    }
```

C++언어

구축순서

C++는 클라스선언시 표시된 순서대로 클라스성원함수의 구축자를 호출하게 하며 자료성원초기화목록에 대하여서는 순서대로 호출하지 않는다. 다음의 Obj클라스선언을 고찰하시오.

```
class Obj{
    public:
        Obj (float xcoord , float ycoord,
        const color &c);
        ...
    private:
```

```
            Position MyPosition;
            color MyColor;
    };
```

Obj구축자의 실현부는 다음과 같다.

```
    Obj::Obj(float x, float y, const color &c):
        MyColor(c), myposition(x, y) {
        //코드는 필요없다.!
    }
```

정의가

```
    Obj BlueObj(3.5, 4.6, Blue);
```

와 같이 될 때 클라스선언시 MyColor전에 MyPosition이 나타나므로 MyPosition구축자는 MyColor구축자보다 먼저 호출된다. 혼돈을 피하기 위하여 클라스선언에서 같은 순서로 자료성원을 초기화해야 한다.

RectangleShape의 그리기함수와 같은 방법으로 코드는 타원의 너비와 높이, 타원의 중심을 리용하여 외접한 4각형의 자리표를 위한 Position객체를 창조한다.

TriangleShape의 그리기성원함수는 복잡하게 동작한다. 3각형을 그리기 위한 낮은 준위루틴은 RenderPolygon을 호출한다. 이 함수는 3개의 변수를 요구한다. 하나는 Position객체의 배렬인데 그리려는 다각형의 정점의 위치이다. 두번째는 배렬에 포함된 정점의 개수이고 세번째는 다각형의 색이다. 이 기능을 수행하기 위하여 TriangleShape의 그리기함수는 3각형의 정점을 구해야 한다. 바른3각형은 같으므로 그리기 쉽다.

3각형의 중심은 세변의 수직2등분선의 사귐점에 놓인다. 그림 13-6은 수직2등분선을 가진 바른3각형을 보여 준다. 3각형의 중심이 자리표 (x,y)로 주어 졌다고 하자. 이 그림에서 보는바와 같이 정점 1의 위치는 (x, y c), 정점 2의 위치는 (x-a, y+a) , 정점 3의 위치는 (x+b, y+a)이다. 물론 b는 3각형의 한변의 길이이다. 다음의 3각형공식을 고찰해 보자.

$$\tan\theta = \frac{a}{b} \quad , \quad \cos\theta = \frac{b}{c}$$

각 θ가 30일 때 a와 c는 다음과 같이 된다.

$$a = \tan 30 \cdot b \,,\quad b = \frac{1}{2} \text{ 이므로 } a = \tan 30 \cdot \frac{1}{2}$$

$$c = \frac{b}{\cos 30} \,,\quad b = \frac{1}{2} \text{ 이므로 } c = \frac{1}{2} \cdot \cos 30$$

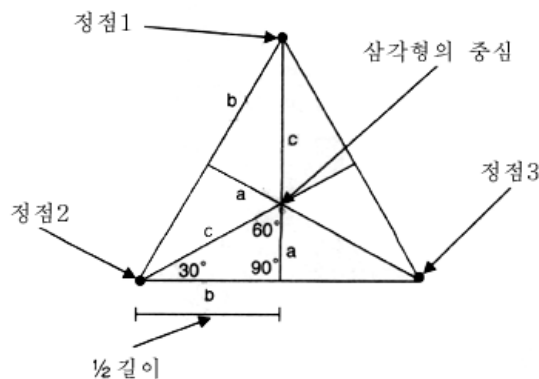



그림 13-6. 3각형과 그의 중심, 정점

바른3각형의 선의 중심과 길이가 주어 진 바른3각형의 정점을 구하는 코드를 쓸수 있다. 그러나 좀 복잡한 문제가 있다. 그것은 5장에서 라디안각을 가진 삼각함수를 취급할 때 구체적으로 보시오. 1° 는 π÷180라디안이다. a와 c의 길이를 계산하는 코드는 다음과 같다.

```
const double Pi=3.1415;
const float Slength= GetSideLength();
float c=Slength/(2.0*cos(30*Pi/180.0));
float a=tan(30*pi/180.0)*.5*SLength;
```

변의 길이와 함께 RenderPolygen을 통과시켜 위치벡토르를 만들수 있다. 벡토르요소는 매 벡토르요소에 값주기하여 초기화하여야 한다. 벡토르의 위치를 만들기 위한 코드는 다음과 같다.

```
Vector<Position> TrianglePoints(3);
TrianglePoints[0]=Center+Position(0, -c);
TrianglePoints[1]=Center
  +Position(-.5*sLength, a);
TrianglePoints[2]=Center
  +Position(.5*sLength, a);
```

4각형을 만들기 위하여 3각형을 포함하는 창문에 통보를 보낼수 있다. 그 호출은 다음과 같다.

```
GetWindow(), RenderPolygon(Trianglepoints, 3, GetColor());
```

TriangleShape의 그리기함수는 다음과 같이 실행된다.

```
void TriangleShape :: Draw(){
    const float pi=3.1415;
    const positionCenter = Getposition();
    const float slength= GetsideLength();
    //3각형의 중심에서 꼭대기정점까지의 거리 c를
    //그리고 바닥의 중심까지의 거리 a를 계산
    float c= slength / (2.0)*cos(30*pi/180.0));
    float a = tan(30*pi/180.0)*.5*slength;
    //3각형의 정점의 위치를 포함하는 묶음을 창조한다.
    Vector <position> Trinanglepoints(3);
    Trianglepoints[0]=center+position(0, -c);
    Trianglepoints[1]=center+position(-.5*slength, a);
    Trianglepoints[2]=center+position(.5*slength, a);
    //3각형을 그린다.
    GetWindow(). RenderPolygon(Trianglepoints, 3, GetColor());
    return;
}
```

새로운 클라스를 검사하기 위하여 코드가 정확히 동작하는가를 보여 주는 세가지 표본에 대한 프로그람을 작성하여야 한다. 프로그람은 창문의 중심에 세개의 표본을 현시하고 바닥을 기준으로 직선을 맞

춘다. 이를 위하여 EzWindows의 SimpleWindow클라스를 리용한다. 표본이 포함되게 될 창문의 전역 선언은 다음과 같다.

```
SimpleWindow TestWindow("TestShapes", 17.0, 7.0,
Position(4.0, 4.0));
```

이것은 화면의 꼭대기에서 4cm, 왼쪽변두리에서 4cm 떨어 진 왼쪽웃구석에 TestWindow라는 창문을 만든다. 창문은 높이 17cm, 너비 17cm이다. 이 창문은 TestShapes라고 표식한다.

EzWindows의 API가 ApiMain()함수를 첫 단계에 호출한다는것을 명심하시오. 간단한 Apimain()함수는 표본을 그리고 구체례화한다. 표본을 창조하는 코드는 아주 간단하다. 아래에 전체프로그람을 보여 주었다.

```
int Apimain(){
    TestWindow.open();
    TriangleShape T(testWindow, position(3.5, 3.5), Red, 3.0);
    T.draw();
    RectangleShape R(testWindow, position(8.5, 3.5), Yellow 3.0, 2.0);
    R.draw();
    EllipseShape E(testWindow, position(13.5, 3.5), Green 3.0, 2.0);
    E.draw();
    return 0;
}
```

EzWindows는 조종이 끝난 상태에서 조작체계로부터 통보를 받으면 완료하기 위하여 ApiEnd()를 호출한다. 검사프로그람에서는 다음의 코드를 보는바와 같이 창문을 완료하기 위하여 탈퇴한다.

```
int ApiEnd(){
    TestWindow.close();
    return 0;
}
```

그림 13-7은 창문의 결과를 보여 준다. 여기서 정확히 수행된 새로운 표본들을 볼수 있다. 이 간단한 검사는 클라스체계구조를 검사하는데는 충분치 않다. 실천에서 이러한 구체적인 검사를 할수 있다.

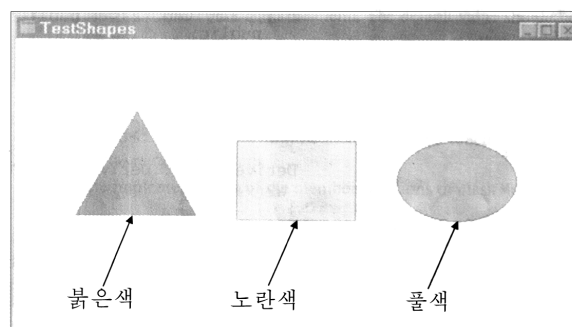



그림 13-7. 시험도형의 현시

문 제

1. 포함을 보여 주는 관계는 무엇인가?
2. 계승을 보여 주는 관계는 무엇인가?
3. 계승에 의하여 다른 클라스로부터 창조된 클라스를 __________클라스라고 한다.
4. 계승체계의 제일 꼭대기클라스를 ____________클라스라고 한다.
5. 다음의 프로그람의 출력은 무엇인가?

```
#include <iostream>
#include <string>
using namespace std;
class weight{
    public:
        wideget(int x);
    private:
        int value;
};
widget::widget(int x): value(x){
    cout <<•widget:"<<value <<end1;
}
class Baseclass {
    public:
        Baseclass(int x);
    private:
        Widget w1;
};
Baseclass::Baseclass(int x): w1(x) {
}
Derivedclass :: Derivedclass(int x): Baseclass(x), W2(x+1) {
}
int main(){Derivedclass B (10);
    return 0;
}
```

6. 아래에 WindowObject의 EzWindow클라스선언을 보여 준다.

```
class WindowObject{
    public:
        WindowObject(SimpleWindow &w,
        const position &p);
        Position getposition() const;
        void getposition(float &x, float &y) const;
```

```
        Simple Window &getWindow() const;
        void Setposition (const position &p);
        void Setposition (float x, float y);
    private:
        SimpleWindow &Window;
        Position Location;
};
```

WindowObject로부터 파생될수 있는 Menu라는 새로운 클라스를 만드시오. 이 클라스는 SimpleWindow내에서 차림표를 만들게 한다. Menu클라스는 다음과 같은 자료성원을 가지고 있다.

- Bitmaps 에 대한 지적자를 동적으로 할당하는 지적자: 이 자료성원 Buttons 를 호출하시오. 비트매프는 차림표의 단추이다.
- 차림표에서 단추의 최대개수를 가리키는 옹근수: NumberOfButtons자료성원을 호출하시오.
- Buttons 의 묶음에서 첨수로 되는 옹근수: 이 옹근수는 단추를 추가하기 위한 다음홈을 지적한다(AddButton 함수아래를 보시오). NextFreeSlot 자료성원을 호출하시오.

WindowObject 로부터 계승된 성원함수에서 Menu 클라스는 다음과 같은 공개성원함수를 가질수 있다.

- 3 개의 인수를 받는 구축자: 첫 인수는 SimpleWindow 를 참조한다. 차림표는 이 창문에 그려 진다. 두번째는 Position 을 참조하는 상수이다. 이 인수는 창문에서 차림표의 위치를 가리킨다. 기정값은 (0.0, 0.0)이다. 세번째는 차림표에서 단추수를 가리키는 옹근수이다. 기정값은 6이다.
- 차림표에 단추를 추가하기 위한 함수: AddButton함수를 호출하시오. AddButton은 하나의 파라메터(Bitmap에 대한 상수지적자)를 가지고 있다. AddButton은 Buttons배렬에 단추를 추가하기 위하여 NextFreeSlot를 리용한다.
- 차림표를 현시하기 위한 함수: Display 함수를 호출하시오. 파라메터는 없다.

Menu클라스의 성원함수에 대한 실현부를 작성하시오.

13.4 보호성원과 계승

C++의 중요한 특징은 클라스의 성원들에 대한 접근을 조종할수 있다는것이다. 앞의 실례에서 보는것처럼 클라스의 성원함수접근을 조종하기 위하여 **public**와 **private**예약어를 리용하였다. 예약어 **protected**는 접근의 3번째 준위를 나타낸다. 기초클라스개념에서 보호성원들은 비공개성원과 같다. 이러한 성원들은 클라스의 성원함수들에 의해서만 호출된다. 실례로 다음의 명령을 볼수 있다.

```
class someclass{
    public :
        void memberFunction();
        int public Data;
    protected:
        int protected Data;
    private :
```

```
            int private Data;};
            void someclass::memberFunction(){
            public Data=1;  //호출가능
            protected Data=2;  //호출가능
            protected Data=3;  //호출가능
    }
    void NonmemberFunction() {
        someclass c;
        c.public Data=1;   //접근가능
        c.protected Data=2;    //접근불가능
        c.protected Data=3;  //접근불가능
    }
```

MemberFunction()성원함수는 3개의 자료성원들에 접근할수 있다. 그러나 비성원함수 NonmemberFunction()는 공개부문에서의 자료성원에만 접근할수 있다. 보호부분과 비공개부분의 자료에는 접근할수 없다. 이 실례를 통하여 보호성원과 비공개성원의 차이점을 알수 있다. 차이점은 새로운 클라스가 기초클라스로부터 공개적으로 파생될 때 명백하다. 아래에 기초클라스와 파생클라스의 선언을 보여 준다.

```
    class Baseclass{
        public:
            int public Data;
        protected:
            int protected Data;
        private:
            int private Data;
    };
    class Derivedclass: public Baseclass{
        public:
            void DerivedclassFunction();
        private:
            //구체적인것은 생략
    };
```

파생클라스의 성원함수는 기초클라스의 공개, 보호성원에 접근하지만 비공개성원에는 접근하지 못한다. DerivedclassFunction()은 기초클라스성원에 대한 파생성원함수의 접근을 보여 준다.

```
    void Derivedclass:: DerivedclassFunction() {
        public Data=1; //접근가능
        protected Data=2;  //접근가능
        private Data=3;  //접근가능
    }
```

C++의 보호기구를 설명하기 위하여 자료성원을 접근하지만 보호기구는 성원함수에게만 적용된다는 것을 강조한다.

기초클라스의 설계에서 보호성원을 리용하는가 하는 문제가 제기된다. 이에 대해서 두가지로 대답을 줄수 있다. 그 대답의 하나는 13.5에서 비공개, 보호부계승을 취급할 때 알수 있다.

표본의 계층구조에서는 보호부문을 리용하지 않는다. 파생된 도형(Rectangle, TriangleShape)은 계승된 공개검토자함수를 통하여 자기의 원래자료성원에 접근한다.

이러한 동작은 파생클라스에서 본래의 클라스자료성원을 은페시킨다. 그러나 일부 경우에 검토자에 의하여 계승된 클라스성원에 접근하는것은 직접 접근하는것보다 품이 더 든다. 이 경우 설계에서 그 기능이 고려되였는가를 알수 있으며 보호자료를 정확히 사용하게 한다.

이런 방법으로 파생클라스의 성원함수는 기초클라스에 의하여 얻어 진 자료에 직접 접근한다. 창문객체클라스의 정의는 다음과 같다.

```
class WindowObject{
    public :
        WindowObject(SimpleWindow &Window, const position &p);
        Position Getposition() const;
        void Setposition(const position &p);
    protected:
        Position center;
        SimpleWindow &Window;
};
```

Shape클라스의 선언은 다음과 같다.

```
class Shape: public WindowObject {
    public:
        Shape(SimpleWindow &Window, const position &p)
        const color &c=Red);
        Color GetColor() const;
        void SetColor(const color &c);
    protected:
        Color color;
};
```

여기서 보는바와 같이 성원자료는 보호부문에서 선언되였다. 이러한 처리는 WindowObject와 Shape로부터 공개적으로 파생된 클라스의 성원함수가 이 자료를 직접 접근하게 한다. 실례로 이 클라스 선언에 의하여 RectangleShape의 Draw성원함수는 다음과 같이 실행된다.

```
void RectangleShape::Draw(){
    const Position UpperLeft = Center
      +Position(-.5*Width, -.5*Height);
    const Position LowerRight = Center
      +Position(.5*Width, .5*Height);
```

```
        GetWindow().RenderRectangle(UpperLeft,
          LowerRight, GetColor());
    }
```

여기서 GetWindow함수는 WindowObject로부터 계승되며 GetColor는 Shape로부터 계승된다. 그러나 이 판본에서 공개검토자함수의 접근은 없어 지고 Shape와 관련한 자료성원은 직접 접근된다. 대체로 직접 접근되는 자료성원은 무시한다. 자료성원에 대한 변화가 모든 성원함수에 영향을 주기때문에 클라스의 동작은 유연하지 못하게 되며 또한 검토자도 정확한 동작을 하지 못한다.

13.5 계승의 조종

보호부분을 왜 리용하는가에 대한 두번째 대답은 사용된 종류로 하여야 한다. 계승의 3가지 형태(공개, 비공개, 보호)에 대하여 시험해 보자.

13.5.1 공개부계승

Shape클라스를 취급한 우의 실례들에서는 공개부계승을 리용했다. 실례로 RectangleShape의 선언은 다음과 같다.

```
class RectangleShape:: public Shape{
    public:
        //구체적인것은 생략
    private:
        //구체적인것은 생략
};
```

이 선언은 Shape의 공개성원이 RectangleShape의 공개성원으로 되며 RectangleShape객체가 Shape의 공개함수로 리용된다는것을 의미한다. 실례로 Shape의 SetColor성원함수가 RectangleShape의 리용자로 공개적으로 리용되므로 다음의 코드가 유효하게 된다.

```
RectangleShape R(Window, P1);
R. SetColor(Green); //Shape의 성원함수리용
R.Draw();
```

공개부계승과 함께 기초클라스로부터 계승된 파생클라스의 성원은 기초클라스에서와 같은 보호를 가지게 된다. 공개부계승은 is-a관계모형이기때문에 실천에서 항상 리용된다.

13.5.2 비공개부계승

비공개부계승(private inheritance)을 설명하기 위하여 RectangleShape의 선언을 고찰해 볼수 있다.

```
class RectangleShape: private Shape{
    public:
        //구체적인것은 생략`
    protected:
        //구체적인것은 생략
```

```
    private:
        //구체적인것은 생략
};
```

접근지정자는 private로 변화된다.

```
RectangleShape R(Window, P1);
R. SetColor(Green);
R.Draw();
```

이 코드는 성립되지 않는다. 왜냐하면 RectangleShape의 사용자는 RectangleShape의 비공개성원이기때문에 Shape의 공개성원함수에 접근하지 못하게 한다. 비공개부계승과 함께 기초클라스의 공개, 보호성원들은 파생클라스의 비공개성원으로 된다. 파생클라스의 리용자는 기초클라스에 의하여 제공된 기능에 접근하지 못한다. 기초클라스의 비공개성원들이 파생클라스의 성원함수들에 접근한다는것은 힘들다.

비공개부계승은 공개부계승보다 자주 리용되지 않는다. 그것은 기초클라스의 기능이 사용자나 파생클라스가 보여 주는 대면부의 부분이 아닐 때 리용된다. 비공개부계승은 사용자로부터 기초클라스를 은폐시키며 대면부사용자를 변화시키지 않고 기초클라스의 실행을 변화시키거나 다 없앤다. 접근지정자가 파생클라스의 선언에서 나타나지 않을 때에는 비공개부계승이 리용된다. 비공개부계승의 리용이 적합한 경우는 드물고 다른 방법으로 같은 효과를 낼수 있기때문에 다른 교과서를 참고하여 이런 내용을 보기로 하고 여기서는 서술하지 않는다.

13.5.3 보호부계승

보호부계승(protected inheritance)에 의하여 기초클라스의 공개보호성원들은 파생클라스의 보호성원으로 되며 기초클라스의 비공개성원은 파생클라스의 비공개성원으로 된다. 보호부계승은 파생클라스의 실행에서 기초클라스의 편의와 능력이 쓸모 있을 때는 적합하지만 파생클라스사용자가 보게 되는 대면부는 아니다. 보호부계승은 공개부계승보다 얼마 리용되지 않는다. 표 13-1은 계승의 3가지 형태가 파생클라스에 접근할수 있는가 없는가 하는 관계를 보여 준다. 접근불가능한 항목(entry)은 파생클라스가 기초클라스성원에 접근할수 없다는것을 보여 준다.

표 13-1. 계승의 형태와 접근관계

계승형태	기초클라스성원접근	파생클라스접근
Public	public protected private	public protected 접근불가능
protected	public protected private	protected protected 접근불가능
private	public protected private	private private 접근불가능

문 제

7. 다음의 표에서 기초클라스성원에 접근하는 파생클라스의 성원함수에 서로 다른 계승접근지정자가 어떤 영향을 주는가를 써넣으시오.

계승지정자	공개성원	보호성원	비공개성원
공개			
보호			
비공개			

8. 다음의 프로그람에서 틀린것은 무엇인가?

```
#include<iostream>
#include<string>
using namespace std;
class widget {
    public:
        widget(int x);
        int Getvalue() const;
    private:
        int value;
};
Widget::Widget(int x): Value(x){
    cout <<"Made Widget :" <<Value<<end1;
}
int Widget::GetValue()const{
    return value;
}
class Baseclass{
    public:
        Baseclass(int x);
        void Print();
    private:
        Widget W1;
};
class Derivedclass: private Baseclass{
    public:
        Derivedclass(int y);
    private
        Widget W2;
};
Baseclass::Baseclass(int x):W1(x){
```

```
}
void Baseclass::Print() {
    cout << w1, getvalue() << end1;
}
Derivedclass::Derivedclass(int x): Baseclass(x),
  w2(x+1){
}
int main(){
    Derivedclass B(10);
    B.print();
    return 0:
}
```

9. 다음의 프로그람은 옳은가? 만일 그렇다면 무엇이 출력되는가?

```
#include <iostream>
#include <string>
using namespace std;
class height{
    public:
        widget(int x);
        int getvalue() const;
    private:
        int value;
};
height::widget(int x):value(x){
    cout <<"made widget :"<<value<<end1;
}
int widget::Getvalue() const{
    return value;
}
class Baseclass{
    public:
        Baseclass (int x);
    protected:
        widget &Getvalue();
    private:
        widget w1;
};
class Derivedclass:protected Baseclass{
```

```
    public:
        derivedclass(int y);
        void print();
};
Baseclass::Baseclass(int x): w1(x){
}
widget &Baseclass::Getvalue(){
    return w1;
}
Derivedclass::Derivedclass(int x): Baseclass(x){
}
void Derivedclass::print() {
    widget w1=Getvalue();
    cout <<"value is" << w1.Getvalue) <<end1;
}
int main(){
    Drivedclass B(10);
    B.print();
    return 0;
}
```

10. 9번 문제에서 비공개부계승을 리용하여 Drivedclass를 변화시키시오. 프로그람이 동작하는가? 왜 그런가?

11. 다음의 프로그람이 옳은가? 그렇다면 무엇이 출력되는가?

```
#include<iostream>
#include<string>
using namespace std;
class widget{
    public:
        widget(int x);
    protected:
        int Getvalue() const;
    private:
        int value;
};
widget::widget(int x): value(x){
    cout << "made widget:" <<value <<end1;
}
int widget::Getvalue() const{
    return value;
```

```
}
class Baseclass{
    public:
        Baseclass(int x);
    protected:
        widget &getvalue();
    private:
        widget w1;
};
class Derivedclass : protected Baseclass {
    public:
        Derivedclass(int y);
        void print();
};
Baseclass::Baseclass(int x) : w1(x){
}
widget & Baseclass::getvalue(){
    return w1;
}
Derivedclass:;Derivedclass(int x): Baseclass(x) {
}
void Derivedclass::print(){
    widget w1= Getvalue();
    cout << "value is"<<w1. Getvalue()<<end1;
}
int main(){
    Derived class B(10);
    B.print()
    return 0;
}
```

13.6 다중계승

도형계층구조에서는 하나의 계승을 리용하였다. 그러므로 파생된 매 클라스는 하나의 어미를 가진다. C++는 다중계승도 지원한다. 이 기술에 의하여 파생된 클라스는 2개 혹은 그이상의 기초클라스를 계승할수 있다. 파생클라스는 모든 어미의 속성과 동작을 계승한다.

다중계승의 개념을 설명하기 위하여 객체지향기술을 리용한 은행업무을 다시 설계하기로 하자. 먼저 Basic라는 기초클라스를 설계한다. 이 클라스는 임의의 계산에 대한 속성과 동작의 기초모임을 교갑화한다. 또한 이 클라스는 평형을 유지하고 업무처리를 기록하며 업무처리를 인쇄하기 위한 기능도 제공한다.

BasicAccount로부터 특수한 계산형태들이 파생된다. 후보계산형태는 Loan, Interest, Checking, Brokerage 등이다. 그림 13-8에 계승체계구조를 보여 준다.

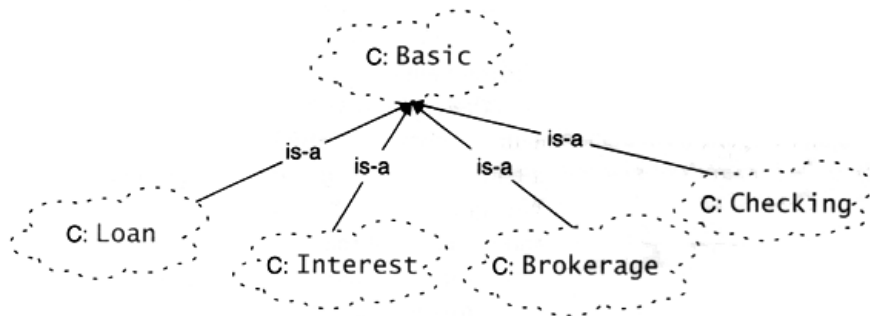



그림 13-8. 기본구좌계승체계구조

현재 씌여 진것을 검사하는 중개구좌나 리자를 지불하는 검사구좌와 같은 구좌형태도 제공한다.이 구좌의 새 형태는 다중계승을 리용하여 창조된다.

InterestChecking구좌는 Checking과 Interest구좌형태와 많은 다른 기능의 파생에 의해 만들어 질 수 있다. 그림 13-9에 새로운 계층체계구조를 보여 준다.

다중계승은 존재하는 2개 클라스의 교차점인 새로운 클라스를 만드는 방법을 제공한다. 실례로 InterestChecking구좌는 Interest나 Checking구좌의 모든 공개성원함수를 불러 낸다. 이처럼 InterestChecking구좌는 Interest와 Checking구좌기능을 둘 다 가진것으로 된다.

다중계승은 새로운 추상을 만드는 강력한 기능도 제공한다. 이 기능은 좀 위험하다. 이것의 능력과 제기되는 일부 문제들을 설명해 주는 창문객체의 체계구조를 확장하기 위하여 다중계승을 리용하자. 이 체계에서(그림 13-4를 참고) 아직 개발되지 않은 Label클라스를 보여 준다. 다중계승을 실현하기 위하여 Label을 설계하고 실행하여 사용할수 있다.

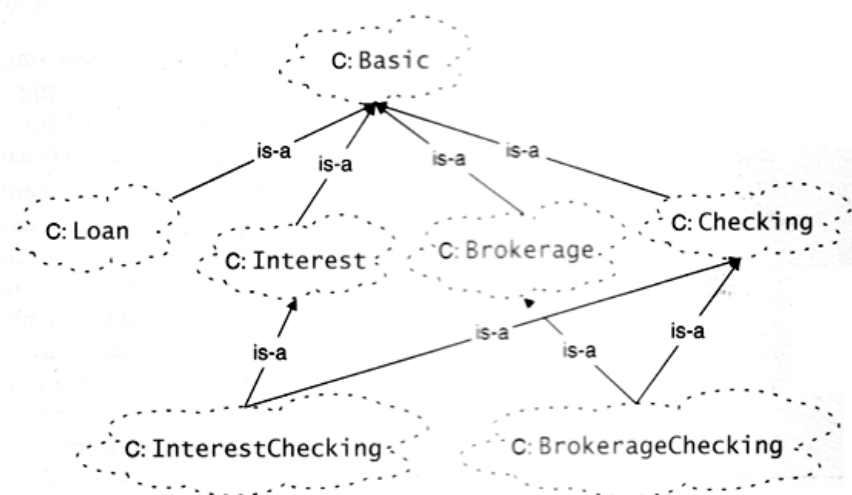



그림 13-9. 다중계승을 가진 구좌체계

Label클라스가 창문에서 본문객체를 배치하는 기능을 제공한다는것을 명심하시오. 그러므로 Label의 비공개자료성원의 하나는 문자렬로 된다. 또한 본문을 현시할 때 화면의 배경색을 지정하는 자료성원도 필요하다. 문자크기와 서체본문의 색을 지정하는 자료성원도 있어야 하는데 아쉽지만 련습은 후에 보

기로 하자.

창문에 나타날 본문을 만드는 성원함수와 비공개자료성원의 변이자와 검토자도 필요하다. Label의 클라스선언은 다음과 같다.

```
class Label: public WindowObject{
    public;
        Label(SimpleWindow &w, const Position &p,
        const string &Text,
        const color  &Color=White);
        color GetColor() const;
        void SetColor(const color &c);
        void Draw();
    private:
        color Color;
        string Text;
};
```

목록 13-6은 Label을 실행하는 프로그람이다. 일부 코드만 제외하고 나머지는 이전에 설명한 표본클라스와 비슷하다. Label구축자는 모든 동작을 수행하기 위하여 성원자료구축자를 리용한다. 성원함수 Draw()는 표본클라스의 그리기함수와 같다. 이 함수는 정확한 위치에 문자를 그리는 SimpleWindow성원함수 RenderText가 요구하는 경계들을 계산한다.

Label은 화면에 객체를 표시할수 있게 한다. 자기의 클라스이름으로 표시하여 표본을 현시하기 위하여 검사프로그람을 변경시켜 보자. 표식과 함께 이 변경은 ApiMain()에 약간의 추가만을 요구한다. 표본을 그린 다음에 표본아래에 놓인 Label객체를 구체례화하고 그것을 현시해야 한다.

목록 13-6. label.cpp에서 Label클라스의 실현부

```
# include <assert.h>
# include "label.h"
Label::Label(SimpleWindow &w, const position &p,
  const string &t, const color &c): WindowObject(w, p),
  Text(t), color(c) {
    //코드가 필요없다.
}
color Label:: GetColor()  const {
    return color;
}
void Label:: Stcolor(const color &c) {
    color = c;
}
void Label:: draw(){
    Position center = Getposition();
```

```
    Position upperLeft = center + position(-2.0, -2.0);
    Position LowerRight = center+position(2.0, 2.0);
    GetWindow(). RenderText(upperLeft, Lowerright, Text, GetColor());
}
```

목록 13-7은 수행된 ApiMain()을 보여 준다. 표식을 추가하기 위하여 창문의 크기와 표본의 크기도 쉽게 변경시킬수 있게 검사프로그람은 파라메터값을 받아 처리하게 되였다. ApiEnd()코드는 변하지 않는다. 그림 13-10은 변경된 프로그람을 실행하게 할 때 창조된 창문을 보여 준다.

목록 13-7. Label클라스에 대한 시험프로그람

```
#include "label.h"
#include "rect.h"
#include "ellipse.h"
using namespace std;
const float Windowwidth = 14.0;
const float WindowHeight=3.0;
SimpleWindow TestWindow ("testShapes",
    Windowwidth, Windowheight, position(2.0, 2.0));
//APImain(): 표본을 그리고 그림들을 표시한다.
int APImain(){
    //표본과 화면테두리사이의 간격을 1cm로 설정
    float Shapewidth=(Windowwidth-4.0)/3.0;
    float ShapeHeight= WindowHeight-2.0;
    position Shapecenter((Shapewidth/2.0)+1.0,
      WindowHeight/2.0);
    position Labelcenter(( Shapewidth/2.0)+1.0,
      (WindowHeight/2.0)+ShapeHeight/2.0+1.75);
    TestWindow.open()
    //3각형을 그리고 표식달기
    TriangleShape T(testWindow, Shapecenter, Red, Shapewidth);
    T.Draw();
    Label Tlabel (testWindow, Labelcenter, " Triangle", white);
    Tlabel.Draw();
    //4각형을 그리고 표식달기
    Shapecenter=Shapecenter+position(Shapewidth+1.0, 0.0);
    RectangleShape R(testWindow, Shapecenter, Yellow,
      Shapewidth, ShapeHeight);
    R.Draw();
    Labelcenter=Labelcenter + positon(Shape width+1.0, 0.0);
    Label Rlabel(TestWindow, Labelcenter, "Rectangle", white);
    Rlabel.Draw();
```

```
    //타원을 그리고 표식달기
    Shapecenter=Shape center+position(Shapewidth+1.0, 0.0);
    EllipseShape e(TestWindow, Shapecenter, Green,
      Shapewidth, ShapeHeight);
    E.Draw();
    Labelcenter=Labelcenter+position(Shapewidth+1.0, 0.0);
    Label Elabel(testWindow, Labelcenter, " Ellipse" , white);
    Elabel .Draw();
    return 0;
}
```

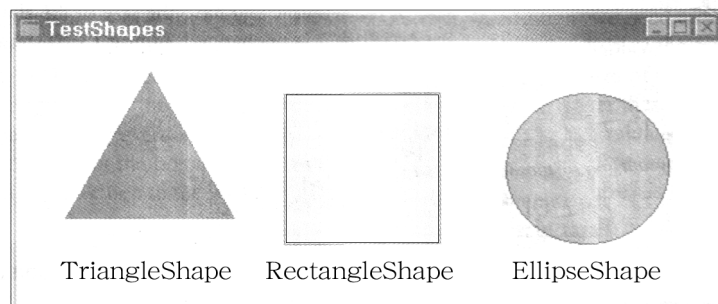



그림 13-10. 시험도형현시

창문객체 Label은 아주 쓸모 있다. 표본에 표식을 달아 주는데 리용할수도 있고 창문에 통보를 내보낼수도 있다. 표식을 포함하는 새로운 형태의 표본을 만들려면 Label을 리용하여야 한다. 실례로 화면에 검은색으로 새로운 표본을 만든다. 화면은 검은색과 흰색이므로 표본의 색을 가리키는 방법이 필요하다. 이 색을 가리키기 위하여 표본가운데 표식을 준다. EzWindows가 지원하는 색은 흰색, 적색, 노란색, 록색, 하늘색, 밤색, 회색이다. 그러므로 표본의 색을 표시하기 위하여 색갈을 표시하는 문자렬의 첫 문자만 리용할수 있다. 규칙적인 표본의 속성과 표식을 가진 새로운 표본형태를 만들어야 한다. 이 새로운 표본을 만들기 위하여 다중계승이 리용된다. 다중계승은 대단히 효과적이고 간단한 방법이다. 적은 코드를 삽입하여 수행할수 있다.

다중계승을 설명하기 위하여 LabeledEllipseShpae라는 새로운 객체를 만들어야 한다. 이 객체는 EllipseShape와 Label로부터 파생된것이다. 그림 13-11에 계승의 계층구조를 보여 준다. 다른 표본의 표식번호는 비슷하게 만들어 질수 있으며 련습에서는 그만두기로 하자.

다중계승을 리용하여 파생된 클라스를 선언하는 문법은 다음과 같다.

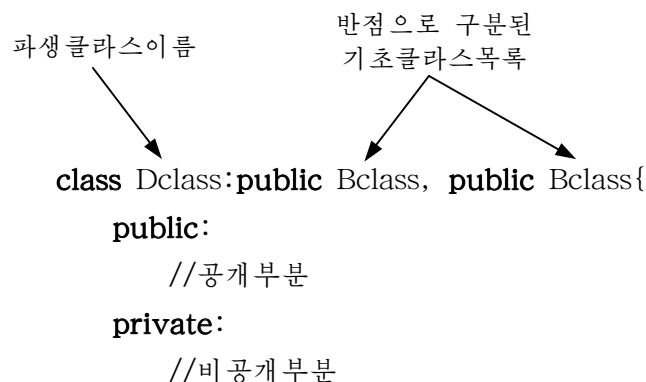
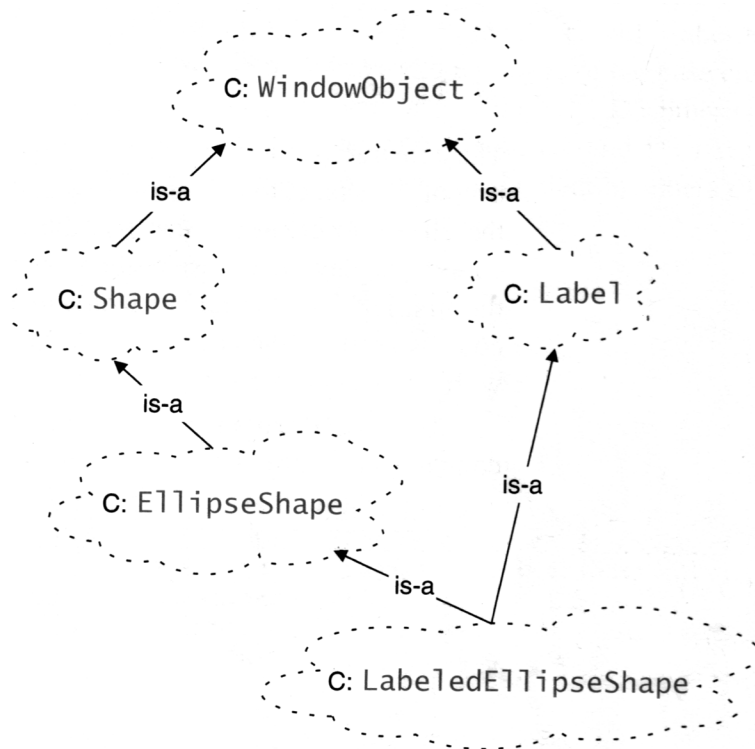



그림 13-11. LabeledEllipseShape의 다중계승의 관계모형

이 문법은 하나의 계승을 지적하는것과 류사하다. 차이점은 하나이상의 클라스가 새 클라스의 기초로 리용된다는것이다. LabeledEllipseShape클라스선언은 다음과 같다.

```
class LabeledEllipseShape : public Label,
  public EllipseShape {
    public:
        LabeledEllipseShape(SimpleWindow &w,
        const position &center, const color &c=Red,
        StringText= " R" float width=1.0,
        float Height= 2.0);
        void Draw();
};
```

LabeledEllipseShape를 위한 구축자는 창문과 표본의 위치, 색, 그것을 표시하기 위한 본문, 만들어야 할 타원의 크기를 지정한다. Draw()는 공개성원함수이다. 물론 그것은 창문에 타원을 현시한다. 타원을 표시하는 기정본문은 타원의 기정색갈에서 첫 문자이다. 자기의 색으로 타원을 표시하는것은 흑백색비데오방식의 콤퓨터에서 실행되는 프로그람인 경우에 아주 유효하다. LabeledEllipseShape의 실현부는 이와 같이 간단하다. 구축자코드는 다음과 같다.

```
LabeledEllipseShape::Labeledellipseshape(
  SimpleWindow &w, const position &center,
  const color c, string t , float Width, float Height) a
  : EllipseShape(w, center, c, Width,Height),
  Label(w, Center, t, c) {
```

```
        //코드필요없음
    }
```

이 구축자가 자료성원을 초기화하는 두개의 기초클라스구축자에 접근한다는것을 알수 있다. 그것들이 모든 동작을 진행하며 구축자구체례내에는 코드가 필요 없다.

LabeledEllipseShape의 Draw()성원함수를 실행하기 위한 코드는 다음과 같다.

```
    void LabeledEllipseShape :: Draw(){
        EllipseShape:; Draw(0;
        Label:: Draw();
    }
```

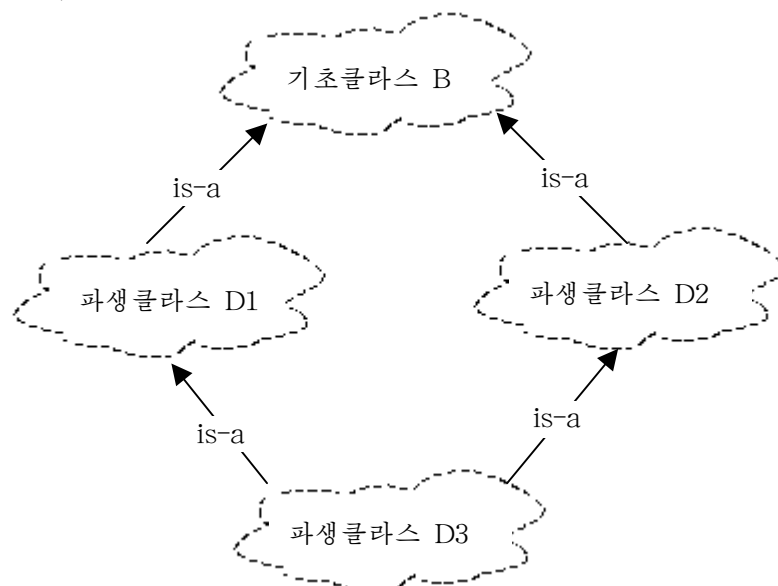
2개의 Draw()성원함수를 호출한다는것을 명심하시오. 다중계승에 의하여 새 클라스가 창조될 때 그것은 기초클라스의 모든 속성과 동작을 계승한다.

따라서 LabeledEllipseShape는 Label과 EllipseShape라는 두 그리기함수를 계승한다. LabeledEllipseShape가 Draw()를 참조할 때 어느것을 불러 내는지 명백치 않기때문에 이러한 상태를 《이름의 모호성》이라고 한다. 이러한 문제를 해결하기 위하여 성원을 지정하는 유효범위해결연산자를 리용한다. 따라서 LabeledEllipseShape를 그리기 위하여 EllipseShape의 타원그리기함수와 Label의 표식을 그리는 함수를 호출한다.

EllipseShape와 Labe1이 같은 클라스 WindowObject로부터 파생되였고 LabeledEllipseShape는 EllipseShape와 Label에서 파생되였으며 LabeledEllipseShape는 WindowObject에서 두가지 구체례를 가지고 있기때문에 이러한 호출은 주의해야 한다.

주의

금강석계승구조

다중계승은 강력한 기능이지만 때때로 여러가지 문제가 제기된다. 다음과 같은 상태가 있다. 하나의 클라스 B에서 파생된 두개의 파생클라스가 있다. 이 클라스를 D1, D2라고 하자. 이 클라스로부터 새로운 클라스 D3을 창조했다. 계승에 의해 파생클라스는 기초클라스의 속성과 성원들을 다 계승한다. D1과 D2는 B의 자료성원을 계승한다. 또한 D3도 D1과 D2의 자료성원을 계승한다. D1과 D2는 B의 자료성원을 가지며 D3은 B의 자료성원이 복사판을 가지게 된다. 직관적인 모형은 다음과 같다.

금강석의 모양과 류사하다는것을 볼수 있다. 의문은 D3에서 B자료성원의 두개의 복사가 실지 요구되는가 하는것이다. 아마 없을것이다. 이 경우에도 지적자와 관련한 흥미 있는 문제가 있다. 일반적으로 다중계승구조에서는 이 경우를 피해야 한다.

2개의 GetWindow()성원함수, 2개의 GetPosition(), 2개의 SetPosition()을 가진다. 또한 비공개성원자료 center와 Window를 가진다. 일반적으로 2중성은 문제가 없지만 개별적성원함수를 불러 내야 한다면 유효범위해결연산자는 적당한것을 선택할수 있다.

LabeledEllipseShape실례에서 보여 주는것처럼 다중계승은 새롭고 강력한 클라스를 손쉽게 창조하게 한다. 코드작성을 얼마하지 않고도 리용하기 쉬운 클라스를 창조할수 있다.

LabeledEliipseShape의 동작을 보여 주는 프로그람을 아래에 제시한다.

```
#include <assert.h>
#include "lellipse.h"
const float WindowWidth = 10.0;
const float WindowHeiht = 3.0;
SimpleWidth, TestWindow("TestShapes"
  Windowidth, WindowHeight, Position(2.0, 2.0));
int ApiMain() {
    float EllipseWidth = (WindowWidth □4.0) / 3.0;
    float EllipseHeight = WindowHeight □2.0;
    position Center((EllipseWidth / 2.0) + 1.0,
      WindowHeight / 2.0);
    TestWindow.Open();
    Assert(TestWindow.GetStatus() == WindowOpen);
    LabeledEllpseShape E1(TestWindow, Center,
      Red, "R", EllipseWidth,  EllipseHeight);
    Center = Center
     + Position(EllipseWidth + 1.0, 0.0);
    LabeledEllipseShape E2(TestWindow, Center,
      Green, "G", EllipseWidth, EllipseHeight);
    E2.Draw();
    Center = Center
      + Position(EllipseWidth + 1.0, 0.0);
    LabeledEllipseShape E3(TestWindow, Center,
      Blue, "B", EllipseWidth, EllipseHeight);
    E3.Draw();
    return 0;
}
```

이 코드는 여러가지 색갈의 타원을 창조하고 현시한다. 타원은 그림 13-10이 보여 주는것과 같이 자

기가 가지고 있는 색의 첫 문자로서 표식되였다.

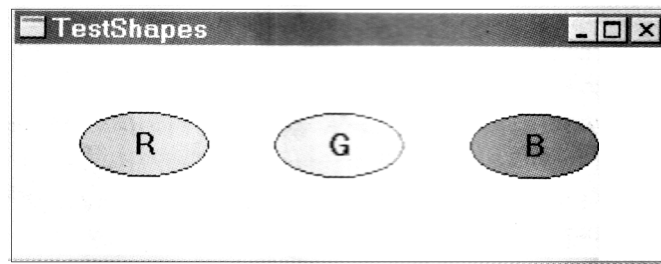



그림 13-12. 표식 붙은 타원

문 제

12. 다중계승을 리용하여 파생된 클라스는 기초클라스의 성원함수와 같은 이름을 가진 성원함수들을 가진다. 성원함수를 호출하는 방법을 설명하시오.
13. 다중계승을 리용하여 EzWindows에서 리용할수 있는 막대기도형을 그리는 bar라는 새로운 클라스를 창조하시오. 매 막대기는 자기의 특성을 가지고 있다.
14. 다음의 클라스선언을 분석하시오.

```
class Obj1 {
    public:
        Obj1 (int v = 0);
        int GetValue() const;
    private:
        int Myvalue;
};
class Obj2 {
    public:
        Obj2 ( int v = 1.0f);
        float GetValue() const;
    private:
        float MyValue;
};
```

파생된 클라스의 선언은 다음과 같다.

```
class Obj3 : public Obj1, public Obj2 {
    public:
        Obj3( int v1, float v2);
        void GetValues(const int &v1,
        const float &v2) const;
    private:
        //아무것도 없다.
};
```

Obj3의 구축자를 실행하는 코드를 작성하시오.
Obj3의 성원함수 GetValues()을 실행하는 코드를 작성하시오. 이 성원함수는 2개의 기초객체 Obj1과 Obj2에 값을 돌려 준다.

13.7 흥미 있는 만화경

동작시키기전에 새로운 도형클라스가 있어야 한다. 새로운 도형을 리용하기 위하여 실지 만화경프로그람을 고찰해 보자. 이 프로그람에서 클라스를 창조하고 리용할수 있는 객체지향프로그람의 우점을 살리기 위하여 프로그람을 다시 설계하기로 하자.

프로그람의 동작에 대한 명백한 지령은 프로그람을 설계하는데 도움을 준다. 이 지령으로부터 프로그람을 구축하는데 필요한 추상이 결정된다. 만화경프로그람의 동작에 대한 지령도 여기에 있다.

만화경프로그람은 창문에 만화경그림을 현시한다. 이 창문은 크기가 10cm인 4각형으로서 Kaleidoscope라고 표식이 되여 있다. 만화경그림은 원, 4각형, 3각형으로 되여 있다. 만화경은 순간에 《변환》하면서 그림을 수시로 변화시킬수 있다. 매번 만화경이 변환되면 같은 형태를 가진 4개의 표본이 그림에 추가된다. 표본의 중심은 원을 4등분한 선우에 놓인다. 표본의 형태와 크기, 색은 매번 우연히 변한다. 대각선을 마주하고 있는 도형은 다 같은 색이다. 우의 명령문에서 창문, 만화경그림 그리고 표본에 대한 추상이 필요하다. 창문에 대한 추상으로써 이전에 소개한 SimpleWindow클라스를 사용할수 있다. 3각형을 이미 가지고 있으므로 계승으로서 4각형과 원을 쉽게 만들수 있다. 그림 13-13은 확장된 표본의 계층구조를 보여 준다.

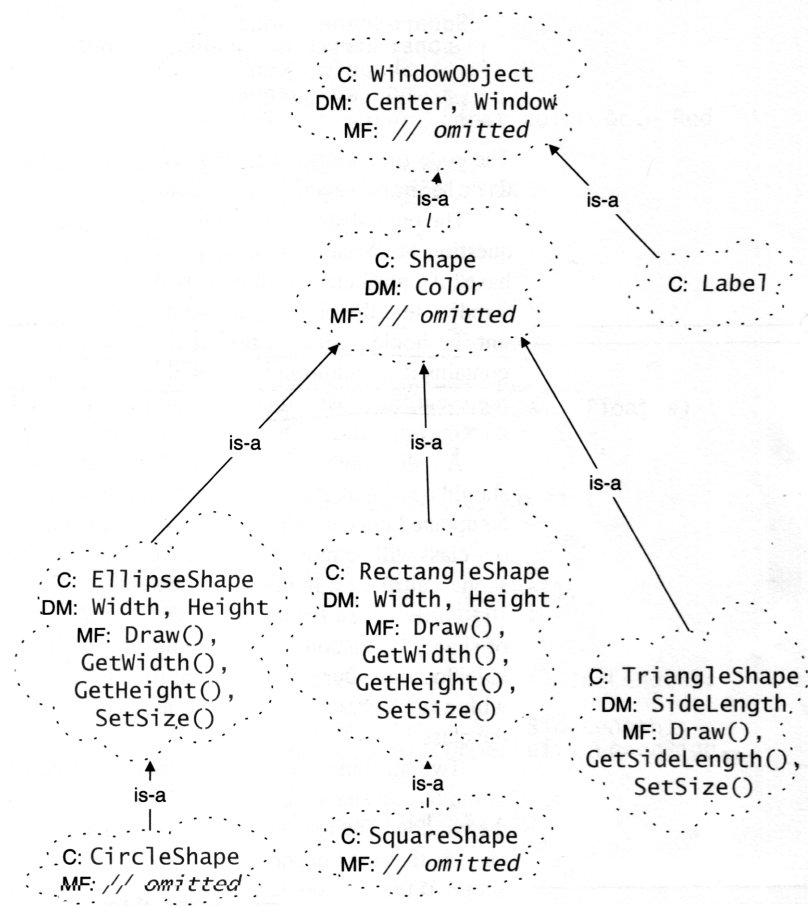



그림 13-13. 확장된 도형의 계층구조

이 계층구조는 좀 복잡하다. 파생클라스가 어미의 모든 속성과 동작을 계승한다는것을 명심하시오. CircleShape는 EllipseShape로부터 공개성원함수 SetSize()와 Draw()를 계승한다. 수행되는 표본구축자와 비슷한 CircleShape구축자가 있다고 하자.

```
Circle C(TWindow, Position(4.0, 4.0), Green, 2.0);
```

은 SimpleWindow TWindow에서 직경이 2cm인 록색원을 만든다. 원의 위치는 창문의 꼭대기와 왼쪽 테두리에서 4cm 떨어 진 곳에 놓인다. 다음과 같이 쓸수 있다.

```
C. SetSize(2.0, 1.0);
C.Draw()
```

이 코드는 원을 그리지는 않지만 큰 직경이 2cm, 작은 직경이 1cm인 타원을 그린다. 문제는 EllipseShape가 CircleShape는 가지지 못하는 특성과 동작을 가지고 있다는것이다. CircleShape는 EllipseShape가 아니면 EllipseShape로부터 CircleShape를 파생할수 없다. SquareShape와 RectangleShape도 이와 같다. 클라스계층구조를 정의할 때 어미의 모든 속성과 동작이 새끼클라스와 꼭 맞는가를 검사해야 한다.

EllipseShape와 RectangleShape로부터 CircleShape와 SquareShape를 파생시키지 않고 Shape에서 그것들을 파생할수 있다. SquareShape의 클라스선언은 다음과 같다.

```
class SquareShape : public Shape {
    public :
        SquareShape(SimpleWindow &w,
        const Position &Center, const color &c = Red,
        float SideLength = 1.0);
        float GetSideLength()  const;
        void SetSize(float SideLength);
        void Draw();
    private:
        float SideLength;
};
```

이것을 실행하는 코드는 다음과 같다.

```
SquareShape::SquareShape(SimpleWindow &w,
    const Position &center, const color &c,float s)
      : Shape(w, center, c), SideLength(s) {
    //코드는 필요없다.
}
```

원을 그리는 코드도 이와 류사하다. SquareShape에 대한 코드와 CircleShape에 대한 코드는 목록 13-8과 13-11에 있다.

설계를 떠난 추상은 만화경을 위한 추상뿐이다. 만화경추상은 어떤 속성과 동작인가? 창문에 표시되는 모든 객체의 속성은 객체와 관련된 창문이다. 만화경클라스도 같다. 그것을 포함한 SimpleWindow의 구체레에 대한 참조가 포함된다. 창문을 추가하여 만화경클라스는 유리보석을 표시하기 위한 객체를

요구한다. 유리보석을 표시하기 위하여 새로 개발한 CircleShape, SquareShape, TriangleShape클라스들을 리용한다.

만화경의 명백한 속성은 그것이 변화될 때의 속도이다. 프로그람에 서술한바와 같이 만화경은 1초에 한번씩 변한다. 클라스에 대해서는 고정된 상수를 써야 했지만 만화경의 속성인 속도를 변화시킨다면 클라스는 더욱 유연하게 동작하게 된다. 여기서는 요구하는대로 값이 변화된다.

만화경추상의 명백하지 않은 다른 속성은 가질수 있는 표본의 형과 만화경의 매 변화를 그리기 위한 표본의 수이다. 목표는 추상에서 만화경의 모든 특징을 얻어 내는것이다. 자료속성은 언제나 클라스의 비공개성원으로 된다.

특별히 추상의 부분이 아닌 2개의 속성은 창조할수 있는 가장 큰 표본의 크기와 창문중심으로부터 가장 큰 편위대한 장군님값이다. 클라스의 이런 속성부분이 생기지 않도록 하는것은 그것들이 만화경을 포함하는 창문의 크기에 의존되기때문이다. 창문이 크다면 더 큰 표본을 리용할수 있지만 창문이 작다면 더 큰 표본을 그릴 필요는 없다. 이것들은 한번 계산되여 클라스에 보관된다고 할수 있지만 창문의 크기를 변화시킨다는것을 명심해야 한다. 이 경우에 만화경을 조절할 필요가 있다. 결과 이런 속성들은 만화경이 변할 때마다 계산되며 이때 창문은 크기가 변한다.

만화경의 속성을 간단히 서술한다.

- 화상을 현시하기 위한 SimpleWindow
- 유리보석을 표시하는 표본들 CircleShape와 SquareShape, TriangleShape
- 만화경의 변화속도

7장에서 개발한 만화경프로그람과 같이 보석을 그리기 위하여 보석의 색갈, 크기 그리고 창문의 중심으로부터의 편위대한 장군님값을 얻어야 한다. 또한 보석의 표본형태를 얻어 내야 한다. 이런 값들을 얻어 내기 위하여 8장에서 개발한 클라스 Random을 리용한다.

목록 13-8. Square.h 에서 SquareShape 클라스의 선언

```
#ifndef SQUARESHAPE_H
#definde SQUARESHAPE_H
#include "Shape.h"
class SquareShape : public Shape {
    public:
        SquareShape(SimpleWindow, &w,
        const Position &center, const color &c = Red,
        float side = 1.0;
        float GetSideLength()  const;
        void SetSize(float SideLength);
        void Draw();
    private:
        float SideLength;
};
#endif
```

목록 13-9. Square.cpp에서 SquareShape의 실현부

```
float SquareShape::GetSideLength() const {
    return SideLength;
}
void SquareShape::Draw() {
    const Position Center = GetPosition();
    float SideLength = GetSideLength();
    Position UpperLeft = Center
      + Position(-.5 * SideLength, -.5 * SideLength);
    Position LowerRight = Center
      + Position(.5 * SideLength, .5 * SideLength);
    GetWindow().RenderRectangle(UpperLeft, LowerRight,
      GetColor());
}
void SquareShape::SetSize(float s) {
    SideLength = s;
}
```

목록 13-10. Circle.h에서 CircleShape클라스의 선언

```
#ifndef CIRCLESHAPE _H
#define CIRCLESHAPE_H
#include "Shape.h"
class CircleShape : public Shape {
    public:
        CircleShape(SimpleWindow &w,
        const Position &Center,  const color &c = Red,
        float Diameter = 1.0);
        float GetDiameter()  const;
        void SetSize(float Diameter);
        void Draw();
    private:
        float Diameter;
};
#endif
```

보석의 형태와 색갈을 얻기 위하여 만화경에 2개의 객체 RandomShape와 RandomColor를 포함시키는데 이것들은 클라스 Random의 구체례들이다.

목록 13-11. circle.cpp에서 CircleShape의 실현부

```
#include "circle.h"
CircleShape::CircleShape(SimpleWindow &w,
    const Position &Center, const color &c, float d)
    : Shape(w, center, c), Diameter(d) {
    //코드는 필요없음
}
float CircleShape::GetDiameter() const {
    return Diameter;
}
void CircleShape::Draw() {
    const Position Center = GetPosition();
    const float Diameter = GetDiameter();
    const Position UpperLeft = Center
      +Position(-.5 * Diameter , -.5 * Diameter);
    const Position LowerRight = Center
      +Position(.5 * Diameter, .5 * Diameter);
    GetWindow().RenderEllipse(UpperLeft,  LowerRight,
      GetColor());
    return;
}
void CircleShape::SetSize(float d) {
    Diameter = d;
    return;
}
```

또한 표본의 형태가 필요할 때 우연히 그것을 주는 RandomShape를 요구한다. 또한 색갈도 RandomColor에 의해 우연히 결정된다.

보석의 크기와 창문에서 그것이 놓이는 위치를 결정하기 위해서 Random 클라스를 리용하였지만 이 값들은 창문의 크기에 의존되는 값들이므로 서로 다른 란수값이 요구된다. 결과 Random클라스의 객체는 표본의 크기와 편위주소를 결정하여야 할 때 창조된다.

만화경추상에 대한 동작을 결정하기가 어느정도 더 힘들다. 이전의 클라스에서처럼 비공개자료에 대한 검토자가 요구된다. 따라서 GetWindow와 GetSpeed함수가 제공된다. 필요한 다른 동작은 변환동작이다. 만화경이 이 통보를 접수하면 화면에 4개의 새로운 표본이 창조되여 추가된다. 이 통보가 만화경을 포함하는 창문클라스에서 나타나기때문에 창문클라스를 호출하는 공개성원함수로 된다.

구축자와 Turn()을 추가하여 클라스 KaleidoScope는 비공개성원함수를 가지게 된다. 이 함수는 표본을 그리는 창문의 중심으로부터 편위를 계산한다. 이 기능은 Turn()을 불러 낼 때 필요하며 그것은 비공개함수이다. 그림 13-14에 만화경객체가 어떻게 작용하는가를 보여 주었다.

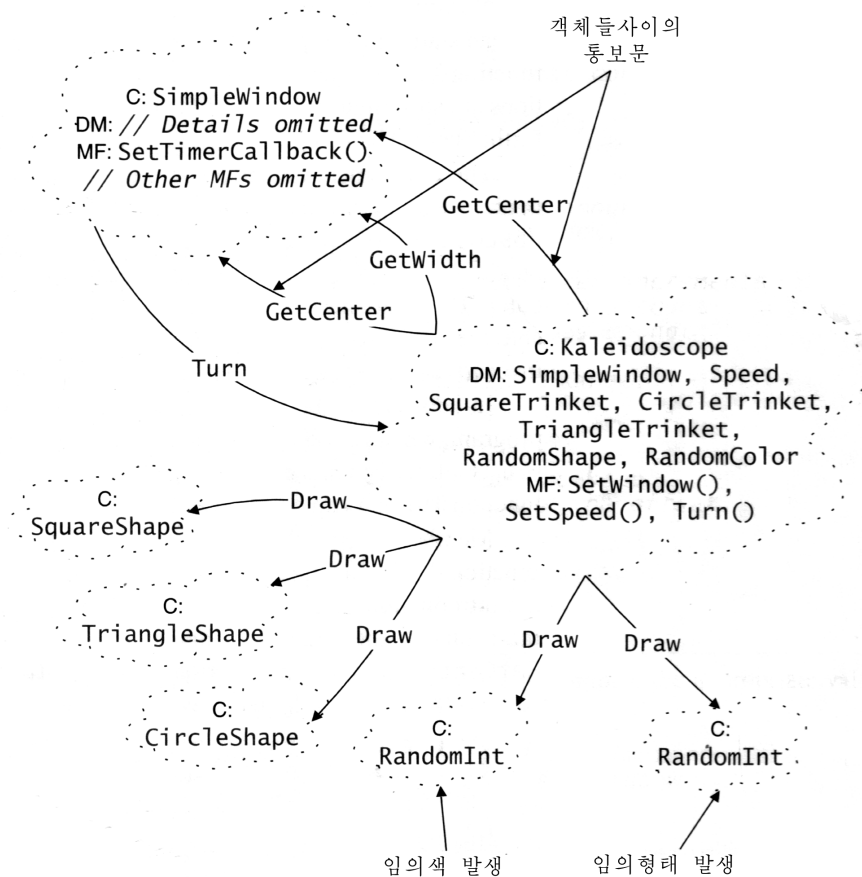



그림 13-14. SimpleWindow와 Kaleidoscope계층구조관계

설계에 기초하여 만화경추상에 대한 클라스선언을 할수 있다. Kaleidoscope클라스선언은 다음과 같다.

```
class Kaleidoscope {
    public:
        Kaleidoscope(SimpleWindow &w, int Speed = 1000);
        int GetSpeed() const;
        SimpleWindow& GetWindow() const;
        int Turn();
    private:
        //형과 상수
        enum { ShapesPerTurn = 4,
        NumberOfShapeTypes = 3};
        enum ShapeType { CircleType, SquareType,
        TriangleType };
        //성원함수
        float RandomOffset(int Range, float ShapeSize);
        //자료성원
        SimpleWindow &Window;
```

```
        int Speed;  //화상을 변화시키는 uSec에서의 속도
        CircleShape CircleTrinket;
        SquareShape  SquareTrinket;
        TriangleShape TriangleTriket;
        Randomint RandomShape;
        Randomint RandomColor;
};
```

우의 선언은 일부를 제외하고는 우리가 정의한 다른 클라스와 류사하다. 클라스선언에서 ShapePerTurn과 NmberOfShapeType를 일체화하기 위해서 일부 대책을 취한다. 클라스선언에서 비공개상수를 정의하려고 한다 해도 C++의 현재판본에서는 초기화된 상수를 클라스의 부분으로 선언하지 못한다. 그대신 렬거형(**enum**)을 리용할수 있다.

```
enum { ShapesPerTurn = 4, NumberOfShapes = 3 };
```

이 선언은 이름을 가지지 않는 렬거형을 선언한다. 그러나 이 렬거형의 성원 ShapePerTurn과 NumberOfShapes는 정의되며 적당한 값을 가진다. KaleidoScope성원함수는 그것들이 상수라면 이 이름을 참조한다.

만화경을 표시하는 창문과 보석을 표시하는 표본은 Kaleidoscope클라스의 자료성원이다. 이 보석형태는 has-a관계를 가진다. 만화경은 창문을 가지며 보석을 가진다. 한편 Kaleidoscope클라스내에서 표본의 형태와 색을 얻어 내는 객체를 교갑화할수 있다.

만화경이 어떤 추상으로 보이는가를 정의하였을 때 동작을 실행해 볼수 있다. 대부분의 작업이 성원함수 Turn()에 의하여 끝나므로 이 실행에 초점을 두어야 한다. 다른 성원함수들은 앞에서 서술한것과 류사하므로 여기서는 그에 대하여 론의하지 않는다. 목록 13-12에 실행코드를 보여 주었다.

다시 설계된 만화경프로그람은 이전의 실행에서 사용되였던 그림을 만드는 방법과 같다. 그러므로 kaleidoscope객체가 변환통보를 받으면 그것은 4개의 표본을 그린다. 대각선에 마주하고 있는 표본들은 같은 색으로 된다. 7장에서의 Turn()성원함수와 만화경함수의 주요기능상 차이는 매번 Turn()을 불러내는것이다. 또한 우연적으로 표본을 그린다는것이다.

이 두 프로그람의 기능상 차이는 또한 실행에서 차이난다. 앞에서 소개한 일부 구축자를 리용하여 성원함수 Turn()의 실행을 더 짧게 할수 있다.

성원함수 Turn()은 만화경함수보다 론리적으로는 더 먼저 시작된다. 그것은 먼저 창문의 중심자리표를 얻는다. 그러나 클라스 Position과 SimpleWindow에 대한 소개와 함께 화상을 포함하는 창문의 중심을 SimpleWindow의 GetCenter()성원함수를 리용하여 얻는데 이때 이 함수는 화상을 포함하고 있는 창문의 중심값을 나타내는 Position객체를 돌려 준다. 이러한 기능을 다음의 코드가 실행한다.

```
//창문중심위치를 얻기
const Position CenterOfWindow =
  GetWindow().GetCenter();
```

판본의 또 한가지 우점은 가장 큰 표본의 크기와 중심위치를 지정하는데서 **const**를 리용하는것보다 이 값을 창문크기함수로 하는것이다.

결과 사용자는 만화경을 포함하는 창문의 크기와 도형의 크기를 다시 변경할수 있으며 그려 진 표본

의 크기를 자동적으로 조절할수 있다. 이 추가된 프로그람의 우연성을 해결하기 위하여 성원함수 Turn()은 알맞는 값을 계산하기 위하여 이 값을 사용하는 Window를 요구한다.

목록 13-12. kaleido.cpp에서 만화경의 실현부

```
#include <assert.h>
#include<iostream>
#include <vector>
#include <string>
#include "kaleido.h"
using namespace std;
//만화경구축자
Kaleidoscope::Kaleidoscope(SimpleWindow &w, int s)
  : Window(w), Speed(s),
CircleTrinker(w, Position(0, 0)),
SquareTrinket(w, Position(0, 0)),
TriangleTrinket(w, Position(0, 0)),
RandomColor(0, MaxColors -.1),
RancomShape(0, NumberOfShapeTypes - 1) {
    Assert(&w  != NULL);
}
//GetSpeed(): 속도를 돌려 준다.
int Kaleidoscope::GetSpeed() const {
    return Speed;
}
//GetWindow(): 만화경을 포함한 창문을 돌려 준다.
SimpleWindow& Kaleidoscope::GetWindow() const {
    return Window;
}
//RandomOfset(): 창문의 중심에서 4각형의 편위위치값을 우연적으로 발생시킨다.
float Kaleidoscope::RandomOffset(int Range,
  float ShapeSize) {
    Randomint R(0, Range *10);
    float Offset = R.Draw() /10.0;
    //발생된 편위위치값이 서로 겹쳐 진 표본을 보관할수 있을 정도로 충분하지 못하다면
    //표본크기의 절반값을 설정
    if (Offset < ShapeSize /2)
        Offset = ShapeSize /2;
    return Offset;
}
```

해당코드는 다음과 같다.

```
//가장 큰 표본은 창문크기의 절반보다 1cm정도 작아야 한다.
float MaxShapeSize = (GetWindow().GetWidth() / 2.0) - 1.0;
//가장 큰 편위값은 창문크기의 절반보다 1cm 작다.
float MaxOffset = (GetWindow().GetWidth()/ 2.0) - 1.0;
```

첫번째 판본에서 표본은 정방형이였고 정확한 방법은 4개의 정방형을 구체례화하고 그것들을 그리는 것이였다. 이 판본에서는 서로 다른 호출을 하게 하며 수행을 간단하게 한다. 그리는 표본의 색과 위치를 얻기 위한 자료구조체를 리용한다. 이러한 방법들은 자료구조체가 그리는 표본의 형태에 관계되므로 이 방법들의 우점은 속성들이 한번 창조되여도 된다는것이다. 다음의 프로그람에서는 2개의 벡토르를 초기화한다.

```
Vector<color> ShapeColor(2);
ShapeColor[0] = (color) Randomcolor.Draw();
ShapeColor[1] = (color) Randomcolor.Draw();
//시간간격 0.1일 때 도형의 크기는 1cm 증가한다.
Randomint RandomShapeSize(10, MaxShapeSize * 10);
const float ShapeSize = RandomShapesize.Draw() / 10.0;
//도형을 그리는 4개의 위치를 계산하고 창조한다.
//창문의 중심위치에서 도형의 편위값을 발생한다.
const float Offset
  = RandomOffset(MaxOffset, ShapeSize);
vector<Position> ShapeLocation(ShapesPerTurn);
ShapeLocation[0] = CenterOfWindow
  +Position( offset, -offset);
ShapeLocation[1] = CenterOfWindow
  +Position( offset, -offset);
ShapeLocation[2] = CenterOfWindow
  +Position( offset, -offset);
ShapeLocation[3] = CenterOfWindow
  +Position( offset, -offset)};
```

벡토르 ShapeColor는 두가지 색을 가질수 있다. 벡토르요소 ShapeColor[0]는 원의 1사분구, 3사분구에 있는 도형의 색을 나타내며 ShapeColor[1]은 2, 4사분구의 도형의 색을 나타낸다. 클라스 Random의 Randomcolor객체는 우연색을 발생한다. 벡토르 ShapeLocation은 4개의 표본을 그리기 위한 위치를 나타낸다. ShapeLocation[0]은 1사분구에 놓이는 도형의 위치이다. ShapeLocation[1]은 2사분구에 놓이는 도형의 위치이다.

그리고 마지막단계에서는 표본을 얻어 내고 4개의 표본을 그린다. 표본을 얻어 내기 위하여 RandomShape라는 다른 Random객체를 사용한다. 그것은 ShapeType에서 정의된 표본의 형태들가운데서 임의의 표본을 얻어 내고 그 표본을 돌려 준다. 마지막단계에서 요구되는 4개의 표본을 그린다. 이것을 실행하는 코드를 아래에 주었다.

```
//그리려는 표본의 종류를 얻기
const ShapeType KindOfShape
  = (ShapeType) RandomShape.Draw();
if (KindOfShape = = CircleType){
    for  (int I = 0; I<ShapesPerTurn; ++i) {
        CircleTrinket.SetPosition(ShapeLocation[I]);
        CircleTrinket.SetColor(ShapeColor[I % 2]);
        CircleTrinket.SetSize(ShapeSize);
        CircleTrinket.Draw();
    }
}
else if (KindOfShape = = SquareType) {
    for (int i=0; i<ShapesPerTurn; ++i) {
        SquareTrinket.setPosition(ShapeLocation[i]);
        SquareTrinket.SetColor(ShapeColor[i % 2]);
        SquareTrinket.SetSize(ShapeSize);
        SquareTrinket.Draw();
    }
}
else if (KindOfShape = = TriangleType) {
    for (int i=0; i<ShapesPerTurn; ++i) {
        TriangleTrinket.setPosition(ShapeLocation[i]);
        TriangleTrinket.SetColor(ShapeColor[i % 2]);
        TriangleTrinket.SetSize(ShapeSize);
        TriangleTrinket.Draw();
    }
}
```

본질적으로 if-then-else명령의 매 갈래는 표본을 하나씩 처리한다. ShapesPerTurn순환의 매 고리는 그려야 할 보석의 속성을 설정한다. 목록 13-12와 13-13은 kaleido.cpp를 실행하기 위한 코드이다.

목록 13-13. kaleido.cpp로부터의 귀환성원함수의 실현부

```
//Turn(): 만화경을 변화시킨다.
int Kaleidoscope::Turn() {
    //창문의 중심에 대한 론리자리표를 얻어 낸다.
    const Position CenterOfWindow =
      GetWindow(). GetCenter();
    //가장 큰 도형은 창문크기의 절반보다 1cm정도 작아야 한다.
    const float MaxShapeSize =
```

```
  (GetWindow().GetWidth() / 2.0)-1.0;
//가장 큰 편위값은 창문크기의 절반보다 1cm 작아야 한다.
const float MaxOffset =
  (GetWindow(). GetWidth() /2.0)-1.0;
//매 분구에 4개의 도형을 창조한다.
//모든 도형은 크기가 갈다. 그러나 크기는 임의로 얻어 진다.
//도형의 색은 임의로 선택된다.
//대각선을 기준으로 서로 마주하는 도형은 갈은 색이다.
vector<color> Shapecolor(2);
ShapeColor[0] = (color) RandomColor.Draw();
ShapeColor[1] = (color) RandomColor.Draw();
//1, 1.1, 1.2…과 같이 도형의 최대크기를 이 간격으로 발생한다.
Randomint RandomShapeSize(10, MaxShapeSize *10);
const float ShapeSize = RandomShapesize.Draw() /10.0;
//도형을 그릴 4개 위치를 만든다.
//창문의 중심위치로부터 도형의 주소를 얻는다.
const float Offset
  = RandomOffset(MaxOffset, ShapeSize);
vector <Position> ShapeLocation(ShapesPerTurn);
ShapeLocation[0] = CenterOfWindow
  + Position( Offset,  -Offset);
ShapeLocation[1] = CenterOfWindow
  + Position(-Offset, -Offset);
  ShapeLocation[2] = CenterOfWindow
  + Position(-Offset,  Offset);
ShapeLocation[3] = CenterOfWindow
  + Position( Offset,  Offset);
const ShapeType KindOfShape
  = (Shapetype) RandomShape.Draw();
if (KindOfShape == CircleType) {
    for (int I=0; I<ShapePerTurn; ++I) {
        circleTrinket. SetPosition(ShapeLocation[I]);
        circleTrinket,SetColor(ShapeColor[I % 1]);
        circleTrinket.SetSize(ShapeSize);
        CircleTrinket.Draw();
    }
}
else if (KindOfShape == SquareType) {
    for (int i=0; i<ShapePerTurn; ++i) {
```

```
                SquareTrinket. SetPosition(ShapeLocation[i]);
                SquareTrinket,SetColor(ShapeColor[i % 1]);
                SquareTrinket.SetSize(ShapeSize);
                SquareTrinket.Draw();
            }
        }
        else if (KindOfShape  = = Triangletype) {
            for (int I=0; I<ShapePerTurn; ++I) {
                TriangleTrinket. SetPosition(ShapeLocation[I]);
                TriangleTrinket,SetColor(ShapeColor[I % 1]);
                TriangleTrinket.SetSize(ShapeSize);
                TriangleTrinket.Draw();
            }
        }
        return 1;
    }
```

프로그람의 나머지부분만을 얻으면 된다. ApiMain()과 ApiEnd()를 작성하여야 한다. Kmain.cpp라는 분리된 모듈에 이러한 함수가 포함된다. 그림 13-15에 프로그람을 작성하는데 요구되는 모듈을 보여 준다.

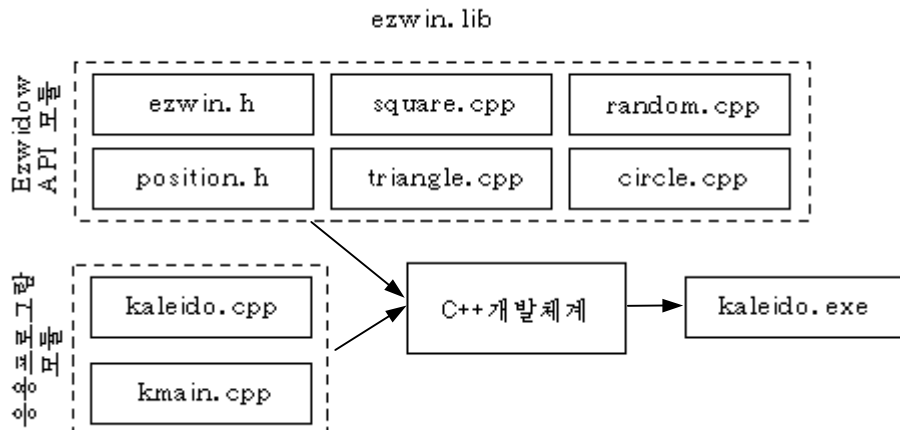



그림 13-15. 만화경프로그람을 만드는데 필요한 모듈

Kaleidoscope객체와 현시되는 창문은 kmain.cpp모듈에서 구체례화된다. Kaleidoscope객체와 창문객체가 프로그람이 실행하는 동안 존재하므로 그것들은 전역객체이다. 객체의 정보는 다음과 같다.

```
SimpleWindow KWindow("Kaleidoscope", 10.0, 10.0,
  Position(2.0, 2.0));
Kaleidoscope Kscope(KWindow, 1000);
```

첫행은 《Kaleidoscope》표식을 단 kWindow라는 창문을 만든다. 창문의 크기는 가로, 세로가 10cm이다. 창문은 화면의 왼쪽웃구석에서 2cm 떨어 진 곳에 위치한다.

두번째 정의는 Kscope라는 kaleidoscope객체를 창조한다. 그것은 KWindow창문에 현시된다. 만화경은 1000ms 즉 1초에 한번씩 변한다. 이 정의순서가 중요하다. 두번째 행은 첫행을 참조한다. C++는 원천파일에서 코드가 나타나는 순서대로 전역선언을 한다. Apimain()은 란수발생기를 초기화하고 KWindow를 열며 KWindow시간계수기에 접속하고 시간계수기를 시동시킨다. 그 코드는 다음과 같다.

```
int DispatchTimerClick() {
    Kscope.turn();
    return 1;
}
int ApiMain() {
    EzRandomize()
    KWindow.Open();
    KWindow.SetTimerCallback(DispatchtimerClick);
    KWindow.StartTimer(Kscope.GetSpeed());
    return 0;
}
```

대체로 코드는 간단하다. 설명이 요구되는 매 단계를 KWindow의 SetTimerCallback성원함수를 위하여 불러 낸다. KWindow가 KScope에 직접 변환통보를 보내기 위하여 이러한 설정을 진행한다. 실례로 다음과 같다.

```
kWindow.SetTimercallback(Kscope.turn);
```

C++의 형검사규칙은 이 명령을 쓰지 못하게 한다. SimpleWindow의 SetTimerCallback함수는 옹근수를 돌려 주는 함수에 대한 지적자를 요구한다. 즉 그것은 int(*)()의 형을 요구한다. 성원함수 Turn()형은 int (Kaleidoscope::*)()이다. 따라서 setTimercallback가 요구하는 파라메터형은 없다. 이 문제를 해결하기 위하여 간단한 간접을 리용한다. KWindow함수는 DispatchTimerClick에 설정하는 시간사건을 호출하며 kscope의 Turn()함수를 호출한다.

프로그람이 완료되기전에 ApiEnd()루틴이 완료하기 위하여 호출된다. 만화경프로그람은 KWindow를 닫고 체계로 돌아 간다. 목록 13-14는 ApiEnd()와 ApiMain()의 실현부를 보여 준다.

목록 13-14. kmain.cpp에서 만화경의 실현부

```
#include "kaleido.h"
SimpleWindow KWindow("Kaleidoscope", 10.0, 10.0,
  Position(2.0, 2.0));
Kaleidoscope KScope(KWindow, 1000);
int DispatchTimerClick() {
    KScope.Turn();
    return 1;
}
int apiMain() {
    EzRandomize();
```

```
        KWindow.Open();
        KWindow.SetTimerCallback(DispatchTimerClick);
        KWindow.StartTimer(Kscope.GetSpeed());
        return 0;
    }
    int ApiEnd() {
        KWindow.Close();
        return 0;
    }
```

그림 13-16은 변화된 만화경그림을 보여 준다. 여기서 보는것처럼 새로운 표본이 추가되여 실제의 만화경과 같이 보인다.

13.8 알아 둘 점

✓ is-a관계는 계승을 나타낸다. 실례로 승용차는 운반수단의 한가지이다. 콜리예는 개의 한 종류이다.

✓ is-a관계는 이행적이다. 시아메스고양이는 고양이의 한 종류이다. 고양이는 포유동물이다. 그러므로 시아메스고양이는 포유동물에 속한다.

✓ has-a관계는 포함관계를 나타낸다. 실례로 라지오는 스위치를 가지고 있다. 승용차는 기관을 가지고 있다. 집합체는 포함을 리용하여 구축된다.

✓ 계승과 포함은 둘 다 쏘프트웨어를 재리용하기 위한 방법이다.

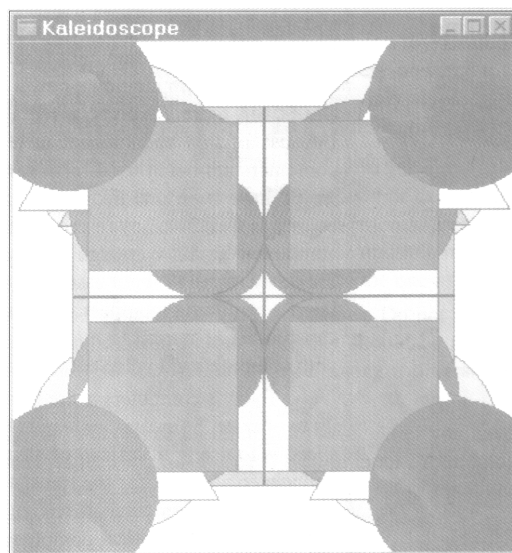



그림 13-16. 개선된 만화경그림

✓ 존재하는 클라스로부터 창조된 새 클라스를 파생클라스, 혹은 부분클라스라고 한다. 어미클라스는 기초클라스 혹은 웃준위클라스라고 한다.

✓ 파생된 클라스의 객체가 구체례로 될 때 기초클라스의 구축자를 파생클라스의 구축자보다 먼저 불러 낸다.

✓ 효과성이 제기되면 .h파일의 클라스의 선언에 실현부가 포함될수 있다. 성원함수는 내부전개식으로

선언될수 있으며 C++콤파일러는 성원함수에 대한 호출을 함수의 본체로 바꾼다. 자료성원초기화목록을 리용하는 클라스자료성원초기화는 구축자로부터 변화를 호출하는것보다 더 효과적이다.

- ✓ 해체자는 구축자의 호출과 반대로 호출된다. 따라서 파생클라스를 위한 해체자는 기초클라스 및 웃준위클라스의 해체자전에 호출된다. 공개부계승과 함께 기초클라스의 공개성원은 파생클라스의 공개성원이다. 기초클라스의 비공개성원들은 파생클라스의 성원함수를 호출할수 없다.

콤퓨터의 력사

기술의 발전

마지막 10년동안은 값 눅고 속도가 빠른 콤퓨터의 도입기간이라고 말할수 있다. 지금 대당 2000딸라인 PC는 1990년대중엽에 대당 40000딸라이상으로 팔리던 콤퓨터보다 80~100배의 계산능력을 가지고 있다. 전문가들은 더욱더 연산속도가 높아 질것이라고 한다. 계산능력의 증가와 가격의 감소로 콤퓨터는 몇해전보다 상상할수 없을 정도로 많은 부분에서 리용된다. 승용차들에 기관을 조종하는 콤퓨터가 들어 가게 되고 사람의 음성을 분석하고 인식하는 콤퓨터가 출현하기 시작했다. 문자를 인식하는 콤퓨터도 출현하였다. 이러한 콤퓨터들이 아직은 그 능력에서 제한은 받지만 음성지령으로 콤퓨터를 조종하는것은 시간문제이다.
지금의 콤퓨터는 대체로 전기를 리용한다. 일부 연구사들은 전기대신에 빛을 리용하여 속도를 높일수 있는 방법을 연구하고 있다. 그들은 트랜스페이저(Transphasor)라는 반도체대용품을 개발하고 있다. 만일 이 반도체가 개발되면 광학콤퓨터는 초당 1조개의 연산을 할수 있다고 한다. 광학콤퓨터는 또 다른 우점이 있다. 광학적리론에 기초한 콤퓨터는 빛묶음이 서로 간섭이 없이 통과하므로 전기에 의한 콤퓨터보다 소형화될것이다. 사실상 전기에 의한 콤퓨터는 서로 교차되지 않는 전기선을 리용해야 한다. 콤퓨터의 급속한 발전은 유기분자를 떼여 놓고는 생각할수 없다. 현재 콤퓨터소편은 규소를 미세한 반도체소편으로 바꾸어 만든것이다. 그러나 규소반도체처럼 유기적인 분자를 만드는것도 가능하게 되였다. 분자준위로 동작하며 더 높은 속도와 작은 체적을 보장할수 있다. 일부 전문가들은 앞으로 규소결정의 콤퓨터를 만드는것보다 단백질에 의한 콤퓨터가 나올것이라고 예측하고 있다.

- ✓ 보호부계승에서 기초클라스의 공개보호성원은 파생클라스의 비공개성원으로 된다. 기초클라스의 비공개성원들은 파생클라스의 성원함수로 호출될수 없다.
- ✓ 다중계승에서 파생클라스는 모든 어미클라스의 속성과 동작을 계승한다.
- ✓ 비공개부계승에서 기초클라스의 공개, 보호성원들은 파생클라스의 성원으로 된다.

련습문제

13.1 놀이감의 계층구조를 설계하시오. 기초클라스는 놀이감이다. 이 놀이감의 부분클라스는 놀이감재질, 전원장치, 기계장치이다. 계승을 리용하여 계층구조를 만드시오. 그림 13-1과 같은 도표를 그려서 계층구조를 설명하시오

13.2 다중계승을 복습하기 위해 련습 13-1의 문제를 다시 풀어 보시오.

13.3 빛에 대한 계층구조를 설계하시오. 실례로 전등빛, 연소빛, 전기발광빛과 같은것이다. 다중계승은 리용하지 마시오. 그림 13-1처럼 도표를 그리고 설명하시오.

13.4 신발의 계층구조를 설계하시오. 다중계승은 리용하지 마시오. 계층구조의 첫 준위는 남자신발과 녀자신발이다. 그림 13-1과 같이 도표를 그리고 설명하시오.

13.5 시계의 계층구조를 설계하시오. 계층구조를 설계한 다음 매 시계형에 따르는 클라스를 창조하시

오. 그림 13-13과 같은 그림을 그리고 클라스선언모임을 정의하시오.

13.6 출판물의 계층구조를 설계하시오. 이 구조를 설계한 다음 매 출판물형에 따르는 클라스를 창조하시오. 그림 13-13과 같은 그림을 그리고 클라스선언모임을 정의하시오.

13.7 콤퓨터인쇄기의 계층구조를 설계하시오. 다중계승을 리용하지 마시오. 그림 13-1과 같은 도표를 그리고 이 구조를 설명하시오. 인쇄기의 형태는 레이자, 잉크분사식, 충격식, 물감인쇄기 등이다.

13.8 기정위치를 가지도록 표본클라스를 변경하시오. 즉 표본을 포함하는 창문만을 제공하여 표본을 선언할수 있다. 코드를 변경하기전에 가장 좋은 기정위치를 결정하시오.

13.9 다음과 같은 클라스구조가 있다.

```
class Top {
    public:
        Top ();
        int look();
    protected:
        int peek();
    private:
        int value;
};
class A:public Top {
    // ...
};
```

파생클라스 A로부터 Top클라스의 어느 함수와 자료성원이 호출될수 있는가?

13.10 클라스 Hoo와 Wahoo를 분석하시오. 3개 행에서 오유를 찾아 내시오.

```
L1      class Hoo {
L2          public:
L3              Hoo ( );
L4              Hoo ( &int n);
L5              int Hoo (int n, int m);
L6          protected:
L7              int HooYear;
L8              int HooTime;
L9          private:
L10             int HooDay;
L11     };
L12     class Sahoo : public Hoo {
L13         public:
L14             Wahoo(int n, int m);
L15     };
L16     Wahoo::Wahoo(int time, int day ) {
L17     HooTime = time;
L18     HooDay = day;
L19     }
```

13.11 클라스 Hounddog는 클라스 dog에서 비공개적으로 계승되였다. Main()에서 정의된 Hounddog 객체 Blue가 다음의 성원을 호출할수 있는가를 말하시오.

1) Hounddog의 비공개성원
2) Hounddog의 공개성원
3) Hounddog의 보호성원

4) Dog의 비공개성원
5) Dog의 공개성원
6) Dog의 보호성원

13.12 화면에 추가되는 도형의 형태를 규정하기 위해 만화경의 Turn()함수는 if-then-else명령을 리용했다. switch명령문은 더 좋은 명령문으로 된다. 이 명령문을 리용하여 코드를 변경하시오. 할수 있는가? switch명령을 리용하는 프로그람으로 변경하기 위해서는 무엇을 변경시켜야 하는가?

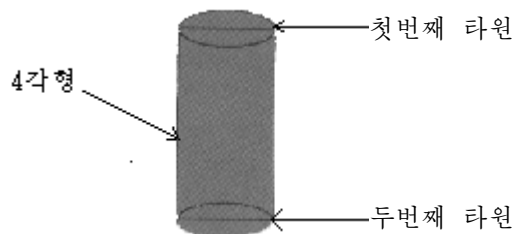
13.13 표본의 위치는 표본의 중심에 주어 진다. 이 방법은 더 많은 표본작업을 하게 하지만 일부 도형들은 처음에 그 중심이 결정되지 않는다. 속박통의 왼쪽웃구석을 리용하여 표본의 위치를 지정하기 위하여 표본의 계층구조를 변화시키시오. 바른3각형에 대한 속박통을 결정해야 한다.

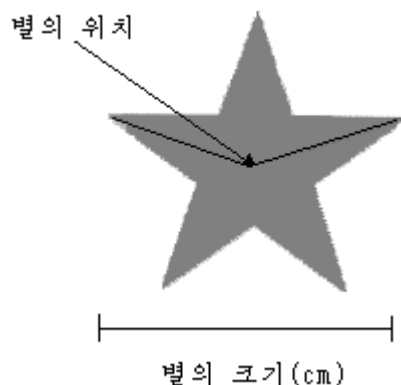
13.14 다음의 표본클라스에 표식을 붙여 주는 프로그람을 설계하고 실행하시오.

1) RectangleShape
2) TriangleShape
3) CircleShape
4) SquareShape

13.15 초기표본계층구조에 다음의 표본을 추가하시오.

1) 금강석형태
2) 5각형
3) 6 각형

13.16 화면에 원통을 그리는 CylinderShape클라스를 작성하시오. 이 클라스는 Shape클라스에서 파생된다. 원통은 두개의 타원과 4각형으로 만들수 있다. 이러한 방법을 리용하여 다음과 같은 원통을 만드시오.

원통의 크기를 파라메터로 받는 프로그람을 작성하여 원통을 그리고 표식을 붙여 주는 CylinderShape의 리용에 대해 설명하시오. 원통의 크기는 cm로 측정한다. 표제의 측정단위는 적당한것을 리용하시오.

13.17 창문에 별을 그리는 StarShape클라스를 창조하시오. 이 클라스는 Shape클라스에서 파생된다. SimpleWindow의 성원함수 RenderPolygon을 리용하여 별을 그릴수 있다. 아래에 starShape의 특징을 보여 주었다.

별의 위치는 중심의 자리표에 의해 결정된다. 별의 크기는 대각선크기에 의해 결정된다. 창문에 6개의 별을 그리는 프로그람을 작성하는데 StarShape의 리용을 설명하시오. 별은 붉은색, 푸른색, 록색, 노란색, 회색, 밤색으로 그릴수 있다.

13.18 StarShape와 지금까지 그린 다른 도형을 리용하여 그 도형들안에 별을 그리는 SimpleWindow를 창조하시오.

13.19 조종창문을 분리하기 위해 만화경프로그람을 변경시키시오. 조종창문은 탈퇴, 전환, 실행, 정지와 같은 단추를 포함한다. 마우스가 탈퇴단추를 누르면 모든 창문이 닫기고 프로그람이 완료된다. 마우스가 절환단추를 누르면 만화경이 변화된다. 만화경이 자동방식에 있다면 절환단추는 비능동으로 된다. 실행단추를 누르면 만화경은 자동방식으로 동작한다. 이 자동방식에서 화상은 1초에 한번씩 변경된다. 만화경이 이미 자동방식에 있다면 실행단추는 필요없다. 정지단추를 누르면 만화경은 수동방식으로 동작한다. 수동방식에서 절환단추가 눌리울 때 화상이 변한다. 만화경이 이미 수동방식에 있다면 정지단추는 필요없다.

13.20 4개의 만화경을 만드는 프로그람을 작성하기 위해서 kaleidoscope클라스를 리용하시오. 화면의 4개 부분에 만화경이 나타난다. 매 만화경은 서로 다른 구역을 절환한다.

13.21 Kaleidoscope클라스가 SimpleWindow에서 파생되므로 만화경프로그람을 다시 설계하시오. 이전에 작성한 프로그람의 우점을 그대로 되살릴수 있는가? 그 리유를 말하시오. 프로그람을 실행시키시오.

13.22 표식된 4각형은(련습 13.14를 보시오.) 단추클라스의 기초로 리용할수 있다. 단추클라스를 설계하고 실행하시오. 단추클라스는 다음의 특징을 가진다. 마우스가 단추내부를 눌렀을 때 사용자정의함수가 호출된다. 이렇게 단추클라스의 자료성원은 함수에 대한 지적자이다. 정지시계를 실행하는 프로그람을 작성하여 새로운 단추클라스를 설명하시오. 정지시계는 시작단추와 정지단추를 가지고 있다. 시작단추가 눌리우면 현재시간이 기록된다. 정지단추가 눌리우면 지나간 시간이 창문에 나타난다.

풀이방향 : 4각형도형을 그리는 IsInside()성원함수를 실행할 필요가 있다.

13.23 계승을 리용하여 새로운 표본형태를 창조하시오. 새로운 견본형태는 그늘도형이다. 실례로 아래에 ShadedRectangleShape를 보여 준다.

Shaded RectangleShape에 추가적으로 ShadedEllipseShape와 ShadedRectangleShape를 창조하시오. 이 클라스들은 도형의 색을 사용자가 지정하도록 한다. 새로운 표본을 리용하는 프로그람을 작성하시오.

13.24 다중계승을 리용하여 새로운 표본형태를 만드시오. 이 표본은 그늘 지고 표시가 되여 있다. 실례로 LabeledShadedRectangleShape는 다음과 같다.

LabeledShadedRectangleShape에 추가적으로 LabeledShadedRectangleShape와 Labeled ShadedTriangleShape를 만드시오. 새 표본을 리용하는 프로그람을 작성하시오.

제14장. 보기와 다형성

소 개

다형성(Polymorphism)은 한 코드표현이 그 코드를 리용하는 객체의 형에 따라 각이한 기능을 하게 하는 언어기구이다. 함수의 다중정의에서 이름의 재리용은 다형성의 기본형식이다. 이 장에서는 다형성을 제공하는 C++의 두가지 기구를 제공한다. 첫번째는 클라스와 함수형이다. 구체적인 형과 값을 불러낼 때 본보기는 새로운 함수, 클라스를 발생할수 있다. 주어 진 형태로 만들어 진 모든 함수와 클라스들은 같은 이름을 가진다. 두번째는 가상성원함수(virtual member function)이다. 가상함수호출에 들어있는 식에서 어느 함수가 리용되는가 하는 결정은 실행시에까지 지연된다(즉 실행시에 가서야 결정된다). 그 결정은 호출에서 참조되는 객체의 형에 기초한다. 이러한 형의 다형성을 흔히 순수다형성(pure polymorphism)이라고도 한다.

기본개념

- 다형성
- 순수다형성
- 함수본보기
- 클라스본보기
- 용기클라스
- 순차목록
- 련결목록
- 2중련결목록
- 반복자클라스
- 클라스에 대한 friend
- 가상함수
- 순수가상함수
- 추상기초클라스
- 가상파생클라스

14.1 범용동작과 형

만일 전문가에게 객체지향언어의 필요한 특성을 물어 보면 그들은 다 같은 말을 할것이다. 그들은 자료와 동작이 하나의 객체로 교갑화되게 하는것이 객체지향언어의 중요한 특징이라고 말할것이다. 클라스와 계승이 그렇다. 객체, 클라스, 계승은 객체지향언어의 기본핵이다. 객체지향언어의 또 하나의 특징은 다형성이다. 다형함수(polymorphic function)는 각이한 형의 객체에 대하여 동작할수 있는 범용함수이다. 취해 지는 동작은 객체의 형에 의존한다. 류사한 동작은 필요는 없지만 요구할수는 있다.

우리가 앞에서 본 다형성의 원시적인 형태는 함수와 연산자의 다중정의(overloading)이다. 다중정의에 의하여 같은 이름을 가진 함수나 연산자를 반복적으로 정의할수 있다. 실례로 앞의 장들에서 Rational과 IntArray연산수로 작업하는 삽입연산자를 다중정의하였다. 다른 실레는 2개의 파라메터를 받아서 더 작은 값을 돌려 주는 Min()함수를 다중정의한것이다.

다중정의와 류사한것이 C++본보기기구이다. 본보기(template)는 다형성과 같은 문법적규칙을 제공한다. 본보기는 개별적요소들이 제공되면 함수나 클라스를 만든다. 어느 함수를 호출해야 하는가 하는것은 실행시에가 아니라 콤파일시에 결정되므로 본보기는 진짜의미에서 다형성과는 다르다.

일부 객체들에 대해서는 본질적인 특성이 콤파일시에 알려 지지 않는다. 그러한 객체들에 대해서는 어느 함수를 호출해야 하는가 하는 결정이 실행시에까지 지연되게 된다. 이러한 기술이 진짜 의미에서의 다형성이다. C++에서는 그것이 가상함수(virtual function)라는 특수한 성원함수를 리용하여 실현된다. 이 장에서 기본고찰대상은 본보기와 가상함수이다.

14.2 함수본보기

C++에는 2가지 종류의 본보기 즉 클라스발생기(class generator)와 함수 및 연산자발생기가 있다. 다음의 코드는 Min()이라는 함수계렬을 생성할수 있는 본보기를 정의한것이다.

```
template <class T>
  T Min(const T &a, const T &b) {
    if (a <b)
        return a;
    else
        return b;
}
```

함수본보기(function template) 혹은 연산자본보기(operator template)의 정의는 예약어 **template**로 시작한다. 이 예약어는 각괄호로 둘러 싼 본보기파라메터목록을 포함한다. 본보기파라메터선언을 반점에 의하여 분리한다. 서로 다른 종류의 본보기파라메터로서 형과 값이 있다. 형본보기파라메터는 어떤 형에 대한 자리유지자를 지정한다. 값본보기파라메터는 어떤 값에 대한 자리유지자를 지정한다. 본보기파라메터목록뒤에는 그 파라메터들을 리용하는 함수에 대한 서술이 놓인다.

값본보기파라메터의 문법은 함수값파라메터와 같다. 즉 형이 먼저 놓이고 그다음 식별자이름이 놓인다. 값본보기파라메터는 대체로 함수본보기에서 리용되지 않는다.

함수본보기에서 형본보기파라메터의 문법을 보면 예약어 **class**가 먼저 놓이고 본보기파라메터의 식별자이름이 뒤에 놓인다.

매개의 본보기파라메터는 함수서술에 리용되여야 한다. 특히 본보기파라메터들은 함수서술의 서명에 리용되여야 한다. 우의 본보기실례에서 발생된 매 Min()함수는 두개의 상수참조파라메터를 가진다. 이 두 파라메터의 형과 되돌림형은 같으며 본보기파라메터 T에서 리용되는 실제형으로 결정된다. 실제형은 본보기이름을 리용하는 함수추출에 의하여 결정된다.

Min()본보기정의와 같은 프로그람에 다음과 같은 코드가 있다고 가정하시오.

```
int Input1;
int Input2;
cin >> Input1 >>Input2;
cout << Min(Input1, Input2) << end1;
```

이때 Min()함수가 자동적으로 발생되며 두개의 상수참조파라메터를 추출한다. 되돌림형은 **int**이다.

```
int Min(const int &a, const int &b) {
    if(a < b)
        return a;
```

```
        else
            return b;
    }
```

이와 류사하게 코드토막

```
    Rational x;
    Rational y;
    cin >> x >> y;
    Rational z=Min(x, y);
```

가 발생하면 두개의 상수 Rational참조파라메터와 Rational되돌림형을 가진 Min()함수가 자동적으로 발생되며 추출된다.

```
    Rational Min(const Rational &a, const Rational &b) {
        if ( a< b)
            return a;
        else
            return b;
    }
```

비교형태가 본보기실체에 의해 달라 지기 때문에 검사표현식 (a < b)는 Min()본보기정의의 흥미있는 부분으로 된다. 추출 Min(Input1 ,Input2)은 **int**비교를 발생시키며 추출 Min(x, y)는 Rational비교를 발생시킨다.

<연산자가 **int**형객체에 대하여 성립하며 Rational서고의 정의에서 보조적인 <연산자를 포함하기때문에 이러한 비교가 발생한다. 다음의 추출은 본보기실체화에서 오유를 발생시킨다.

```
    cout << Min(7, 3.14); //무효: 파라메터형 틀림
```

Min()본보기는 값 7과 3.14가 서로 다른 형이기때문에 적용할수 없다. 본보기에 의해 발생되는 함수에 대하여 번역기는 추출에서 본보기파라메터정의와 실제파라메터의 형으로 정확히 얻어야 한다. 본보기의 지정된 동작이 추출에서 주어 진 형의 객체를 정의하지 않는다면 함수본보기실체화는 오유로 될수 있다. 함수를 발생시키는 시도에서 변환이 자동적으로 되지 않는다. 본보기의 지정된 동작이 추출에서 주어 진 형의 객체를 정의하지 않는다면 함수본보기실체화는 오유로 될수 있다. 실례로 아래의 Min()함수의 추출은 오유로 된다.

```
    cout << Min(cout, cerr); // 무효: 이 방법에서 흐름은 리용될수 없다.
```

<연산자가 흐름의 객체에 정의되여있지 않기때문에 본보기는 정의할수 없다. 함수본보기는 분류에서 효과적인 동작을 한다. 본보기로서 분류알고리듬을 실행하여 임의의 보충적인 동작을 하지 않고도 모든 형태의 배렬에 대하여 분류할수 있다. 아래의 코드는 9장에서 정의한 InsertionSort()의 본보기이다.

```
    template <class T>
    void InsertionSort(T A[], int n) {
        for (int i=1; i<n ; ++i) {
            if (A[i] < A[i-1]) {
```

```
                T v = A[i];
                int j = i;
                do {
                    A[i] = A[j-1];
                    --j;
                }while ((j>0) && (v < A[j-1]));
                A[j] =v;
            }
        }
    }
```

이 본보기정의에서 본보기파라메터 T는 지정되는 객체형이다. 실례로 최소값을 얻는 본보기에서와 같이 연산자 <가 정의되여야 한다. 본보기파라메터는 T함수안에서 리용된다.

함수본보기의 리용에서는 같은 이름으로 함수들을 정의하지 못한다. 프로그람 14-1은 이런 점에 대하여 설명하였다.

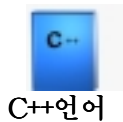
```
#include <iostream>
#include <string>
using namespace std;
template<class T>
void f(T i) {
    cout << "template f(): " << i << endl;
}
void f(int i) {
    cout << "explicit f(): " << i << endl;
}
int main() {
    f(1.5);
    f(1);
    f('a');
    return 0;
}
```

프로그람 14-1. 명시적인 함수정의가 본보기정의를 재정의할수 있다는것을 보여 주는 실례

이 프로그람은 함수본보기 f()와 int형파라메터를 가지는 명백히 정의된 함수 f()를 정의한다. 프로그람결과는 다음과 같다.

```
template f():1.5
explicit f(): 1
template f():a
```

이 실례는 추출함수가 선택되여 있다면 명백한 정의가 본보기정의보다 먼저 처리된다는것을 보여준다.

C++언어

본보기함수와 표준본보기서고

여러가지 자료형의 최대최소값을 계산하는 일이 자주 제기되기때문에 STL알고리듬서고요소는 이러한 작업과 검색, 분류, 결합, 복사, 재배치와 같은 프로그람작업을 위하여 본보기정의 min()과 max()를 제공한다. 알고리듬서고의 기본요소에 대한 목록은 부록 B를 참고하시오.

14.3 클라스본보기

본보기기구는 클라스본보기를 정의하는데 리용될수 있다. 클라스본보기에서 예약어 **template**와 본보기파라메터선언은 클라스서술보다 앞에 놓인다.

함수본보기와 마찬가지로 클라스본보기는 형과 값본보기파라메터를 가질수 있다. 형과 값본보기파라메터는 자료성원의 정의나 성원함수의 표시에 리용된다. 클라스본보기로부터 클라스를 발생시키기 위하여 본보기파라메터를 위한 실제파라메터를 제공한다.

실제파라메터는 본보기이름다음에 놓이며 괄호안에 들어 가 있다. 실제파라메터는 형을 알아야 하며 또한 실제값파라메터는 콤파일시에 계산될수 있는 상수가 되여야 한다. 아래에 TC라는 본보기클라스를 정의하였다. 클라스는 2개의 본보기파라메터 X와 n을 가진다. X는 형본보기파라메터이고 n은 **int**형값 본보기파라메터이다.

```
template<class X, int n>
  class TC{
    public:
        TC();
        void Assign(X xvalue);
        // ...
    private:
        X ValueArray[n];
};
```

본보기클라스 TC<X, n>은 성원함수 Assign()을 가지는데 이 함수의 파라메터 xvalue의 형은 본보기에서 클라스가 만들어 질 때 결정된다. 본보기클라스 TC<X,n>은 배렬자료성원 ValueArray를 가지는데 이 성원의 형과 크기는 클라스가 만들어 질 때 결정된다.

TC<X,n>본보기는 2개의 클라스 A, B를 만들기 위해 객체의 다음정의에서 리용된다.

```
TC<char, 80> A;
TC<int, 125> B;
```

객체 A는 TC<**char**, 80>형이며 **char**파라메터 xvalue를 가진 Assign()성원함수를 가진다. 객체는 자료성원 ValueArray를 가지는데 이 성원은 80개의 **char**요소배렬이다. 객체 B는 TC<**int**, 125>형이며 **int**형파라메터값을 가진 성원함수 Assign()을 가지고 있다. 이 객체는 자료성원 ValueArray를 가지는데 125개의 **int**요소배렬로 되여 있다. 11장에서는 옹근수목록을 표시하는 간단한 용기클라스 IntArray를 개발하였다. 앞으로 여러 부분에서 두개의 더 일반적인 용기를 개발한다. 이 용기들은 서로 다른 능률을 가진다. 이것들을 개발하면 STL의 본보기용기클라스를 쉽게 리해할수 있다.

14.4 클라스본보기를 리용한 목록클라스

어느 용기클라스를 리용해야 하는가 하는것은 목록처리요구에 의존한다. 개별적인 용기클라스를 선택하는데서 쏘프트웨어전문가는 필요한 용기의 특성을 표시하고 결정하는 정보를 검사해야 한다. 프로그람전문가가 용기클라스를 만들 때 제기되는 문제를 아래에 주었다.

- 목록의 최대요소수가 주어 졌는가?
- 목록의 모든 요소들이 같은 형을 가지는가?
- 목록의 요소들을 순서대로 놓아야 하는가? 그렇다면 어떻게 해야 하는가?
- 목록의 요소들이 임의방식으로 혹은 순차방식으로 접근되여야 하는가?
- 요소들의 검사, 변경, 추가, 삭제가운데서 요구되는 기능은 무엇인가?
- 목록이 련관적인가? 즉 열쇠값을 리용하여 목록에 대한 다른 정보를 결정할수 있는가?

먼저 vector와 같은 배렬류사대면부(arraylike interface)를 가진 ADT용기클라스를 만들어 보자. 그후에 STL용기클라스 list와 같은 방법으로 동적요소의 삽입, 삭제를 제공하는 순차접근대면부를 가진 ADT용기클라스를 만든다. 이 두 ADT는 주어 진 집합체의 모든 요소가 같은 형으로 될것을 요구한다. 따라서 동종용기클라스(homogeneous container class)를 만들어야 한다. 이 장에서는 또한 가상함수를 통하여 다형성을 고려할 때 이종목록(heterogeneous list)을 만든다.

ADT배렬류사용기(arraylike container)를 개발하자면 두가지를 선택하여야 한다. 즉 매개의 개별적인 형에 대한 ADT클라스(그에 대하여 intArray와 같은 목록류사(listlike)능력을 주려는)를 개발할수 있으며 아니면 하나의 본보기클라스 ADT를 개발할수 있다. 후자의것이 더 좋다는것은 분명하다.

처음에 개발하려고 하는 용기 ADT대면부가 표준배렬대면부와 같으므로 요소에 대한 접근은 성원첨수지정연산자로 한다. 이것은 또한 다른 기능도 포함한다.

- 임의의 파라메터넘기기방식을 리용하여 그의 객체들중 하나를 넘기기
- 그의 객체들중 하나를 값주기의 원천이나 대상으로 리용하기
- 임의의 파라메터넘기기방식을 리용하여 그의 객체들중 한 객체의 어떤 요소를 넘기기
- 그의 객체들중 한 객체의 어떤 요소를 값주기의 원천이나 대상으로 리용하기
- 그의 객체들중 하나에 의하여 표현되는 요소들의 수에 대한 접근을 제공하기

배렬류사객체들중 하나에 의하여 표현되는 요소들은 형구별(type distinguishing)을 가진다(즉 int 요소들의 모임을 표현하는 목록은 char형요소들의 모임을 표현하는 목록과 다른 형을 가진다). 우리는 또한 목록에 표현되여 있는 요소들의 수가 형구별을 가지는가 하는것도 결정해야 한다(실례로 10개의 int형요소들을 표현하는 객체는 20개의 int형요소들을 표현하는 객체와 다른 형을 가지는가?). 만일 요소들의 수가 형구별을 가진다면 그 요소수는 ADT목록의 값본보기파라메터로 되여야 한다. 요소수가 형구별을 가지지 않는다면 요소수는 목록구축자의 파라메터로 될수 있다.

목록 14-1. Bunch.h에서 클라스본보기 Bunch의 정의

```
template<class T, int n>
  class Bunch {
    public:
        Bunch();
```

```
        Bunch(const T &val);
        Bunch(const T A[n]);
        int size() const { return NumberValues; };
        const T& operator[](int i) const;
        T& operator[](int i);
    private:
        T Values[n];        //목록의 요소들
        int NumberValues;//목록의 크기
}
```

14.4.1 명세부

목록 14-1은 요소수가 형구별을 가지는 Bunch<T, n>이라는 클라스본보기의 대면부를 정의한다. 실례로 아래의 코드토막에서 정의된 A와 B는 서로 다른 형을 가진다.

```
Bunch< int, 10>A;// 10개의 옹근수표현
Bunch<int , 20> B;// 20개의 옹근수표현
```

수행하려고 하는 매 값주기에서 값주기연산자를 명백히 다중정의하지 않는다면 A에 대한 B의 값주기가 성립하지 않는다.

```
A=B; // 무효: A와 B는 다른 형이다.
```

목록 14-2에서는 요소수가 형구별을 가지지 않는 Array<T>클라스본보기를 정의하였다. 다음의 코드토막에서 Array<int>객체 C와 D는 같은 형이다.

```
Array<int> C(10, 1);//10개짜리 하나를 표현
Array<int> D(20, 2);//20개짜리 두개를 표현
```

C와 D가 같은 형이므로 그것들은 서로 값주기할수 있다(값주기의미는 아직 정의되여 있지 않다).

```
C=D;// 옳다:C와 D는 같은 형이다.
```

본보기 Array<T>의 정의는 Bunch<T, n>의 정의보다 좀 더 복잡하다. 왜냐하면 Array<T>객체의 자료성원 Value는 동적공간에 대한 지적자이기때문이며 또한 Bunch<T,n>의 자료성원 Values는 n개 요소들의 배렬이기때문이다. Array<T>객체가 동적공간을 확보하기때문에 Array<T>클라스는 복사구축자와 값주기연산자, 해체자를 정의하여야 한다.

이러한 자료성원들은 Bunch<T, n>객체가 동적공간을 확보하지 않기때문에 Bunch<T,n>을 명시적으로 요구하지 않는다. 즉 콤파일러가 제공하는 복사구축자, 값주기연산자, 해체자이면 충분하다.

목록 14-2.　　　　　Alist.h에서 본보기클라스 Array의 정의

```
template<class T>
  class Array {
    public:
        // 기정구축자는 n개 요소를 val로 초기화한다.
```

```
        Array(int n = 10 , const T &val = T());
        // 표준배렬을 리용하는 구축자
        Array(const T A[], int n);
        // 복사구축자
        Array(const Array<T> &A);
        // 해체자
        ~Array();
        // 목록의 크기를 얻는 함수
        int size() const { return NumberValues; }
        // 값주기연산자
        Array<T> & operator = (const Array<T> &A);
        // 상수목록에서 요소를 찾는 연산자
        const T& operator[](int I) const;
        // 비상수목록에서 요소를 찾는 연산자
        T& operator[](int I);
    private:
        // 성원자료들
        int NumberValues;// 목록의 크기
        T *Values;// 목록요소들에 대한 지적자
};
```

프로그람작성의 유연성은 클라스 Bunch<T,n>이나 Array<T>중 어느것이 ADT의 기초로 되는가를 결정하는데서 기초로 된다. 그러므로 목적은 표현되는 목록요소의 수가 구별되는 형이 아닌 Array<T>를 만드는것이다. 이 절의 나머지부분에서는 클라스 Array<T>의 일부 성원함수와 연산자를 실행하는 방법을 보여 준다.

Array<T>기정구축자는 요소의 수와 요소의 초기값을 지정하는 파라메터 n과 val을 가진다. n의 기정값은 10이고 val의 기정값은 구축된 객체 T의 값이다. 실례로 Rational객체의 목록 R와 문자렬객체의 목록 S가 이미 구축되였다.

```
Array<Rational> R;
Array<string> S;
```

R는 매 요소가 관계형값 0/1을 표시하는 10개의 요소로 된 목록이며 S는 매 요소수가 빈 문자렬을 표시하는 10개 요소로 된 목록이다. Rational클라스의 기정구축자가 0/1의 값을 가지기때문에 R의 요소들은 0/1값을 가진다. 또한 S의 요소들은 string클라스의 기정구축자가 빈 문자렬을 표시하기때문에 " "을 표시한다.

공개기정구축자를 가지지 않는 클라스의 클라스형요소목록을 실체화해야 한다면 n과 val의 값을 명백히 넘겨 받아야 한다. 실례로 RectangleShape클라스는 기정구축자를 가지지 않는다. Array<RectangleShape> 객체를 기정구축한다면 오유통보가 발생한다.

```
Array<RectangleShape> T; // 무효;
```

그러나 아래의 Array<RectangleShape>정의는 성립된다.

```
SimpleWindow W("Drawing window", 20, 20);
RectangleShape r(W, 2, 2, Blue, 3, 4);
Array<RectangleShape> U(5,r); // 5 rs
```

new의 정의는 성립한다. 왜냐하면 요소의 수와 이 요소의 값이 정확하게 정의되였기때문이다.

기본형이 클라스형이 아니고 기정구축자를 가지지 않는다고 하여도 이러한 형의 Array목록을 기정구축할수 있다. 요구가 자기형을 기본으로 하는 객체를 기정구축하는것일 때 콤파일러는 값 0을 자동적으로 주기때문이다. 실례로 다음의 정의는 모두 정확하다.

```
Array<int> V;//10 을 0s 로 표현한다.
Array<double> X;//10을 0.0s로 표현한다.
Array<int> Y(6);//6을 0s로 표현한다.
Array<float> Z(21);//21 을 0.0s 로 표현한다.
```

11장의 IntArray정의와 같이 목록 14-2에서 클라스 Array<T>의 정의는 첨수연산자를 두번 다중정의한다. 2개의 성원첨수연산자는 수식자 **const**의 리용과 되돌림형에서 차이난다. 수식자가 성원함수기호의 한 부분이기때문에 콤파일러는 Array<T>객체에 대한 첨수연산자의 호출을 번역할 때 문맥을 결정한다.

수식자 **const**를 리용한 첨수연산자의 정의는 **const** Array<T>객체가 검사되는 곳에 불러 낸다. 아래에 그 실례를 보여 준다.

```
const Array<int> A(30, 0); // 30 을  0 s 로 표현한다.
cout << A[2] << endl;
int I = A[6];
```

되돌림형을 참조하는 첨수연산자는 비상수형Array<T>객체가 검사되고 변화되는 곳에서 리용된다. 그 실례를 아래에 보여 준다.

```
Array<int> B(10, 1);// 10 을  1 s 로 표현한다.
Array<int> C(20, 2);// 20을  2 s로 표현한다.
B[9] = 17;
Swap(C[3], B[4]);
cin >> B[5];
cout << C[19];
```

클라스 Array<T>의 다른 성원함수 즉 복사구축자, 지정된 구축자, 해체자, 값주기연산자와 size()함수는 IntArray클라스의 일치하는 성원함수에 대하여 같은 목적을 준다. Array<T> 자료성원에 대하여 NumberValues는 목록의 크기를 가지며 Values는 목록의 요소를 가지는 동적기억기에 대한 지적자이다.

14.4.2 실현부

Array<T>성원함수본보기의 정의를 목록 14-3에 보여 준다. 구체적으로 이 목록은 2개의 Array<T> 구축자의 본보기를 제공한다. 클라스정의의 바깥에서 정의된 본보기성원함수에 대한 문법이 좀 애로로 된다(클라스정의내에서 정의된 size()성원은 이러한 방법에서는 애로로 되지 않는다).

같은 파일에서 클라스정의로서 성원함수본보기를 정의한다는것을 명심하시오. 현재콤파일러에 제한성이 있기때문에 정의는 갈라 진 파일에 있을수 없으며 사용자응용프로그람에 련결될수 없다. 콤파일러에 제한성이 있는 근본리유는 Array<T> 그자체를 실행하지 않고 본보기를 실행하기때문이다.

2개의 파라메터 n과 val을 가진 Array<T>기정구축자의 실현부는 n값의 선언으로부터 시작된다. 간단히 말하여 **if**명령문대신 assert()를 호출한다. 만일 n이 값을 가지면 자료성원 NumberValues는 그 값으로 설정된다.

```
assert(n>0);
NumberValues = n;
```

그다음 공간이 목록을 구성하는 값을 유지하기 위하여 확보된다.

```
Values = new T [n];
assert(Values);
```

표준C++에서 이러한 요구를 만족시키지 않는다면 례외가 발생하며 프로그람이 완료된다. 이전의 C++는 이런 불만족한 요구에 값 0을 돌려 주었다. 프로그람실행시에 **for**명령문이 동작하게 되면 충분한 빈 기억기요구부분이 있어야 한다.

```
for(int i=0; i< n; ++i) {
    Values[i] = val;
}
```

for순환에서 Values가 지정하는 곳에 동적공간을 만드는 객체는 val의 값으로 설정된다. 목록 14-3의 두번재 Array<T>구축자는 Array<T>객체를 초기화하기 위한 정확한 표준 C++배렬을 리용한다. 이 구축자의 실체는 기정구축자의 실체와 류사하다. 차이점은 동적공간을 초기화하기 위하여 리용되는 값이다.

Array<T>복사구축자의 본보기, 해체자, 값주기연산자를 목록 14-4에 보여 주었다. 실행은 정확히 되며 IntArray의 일치한 성원함수와 거의 같다. 구체적으로 말한다면 Array<T>복사가 구축이나 값주기를 통하여 만들어 질 때 연산은 지적자 Values의 값을 드문히 반복하는 얕은 복사가 아니라 깊은 복사이다. 즉 갈라 진 목록은 매 요소를 개별적으로 복사하여 만든다.

목록 14-3. alist.h에서 Array구축자본보기

```
// 기정구축자는 n개 요소들을 val로 초기화한다.
template<class T>
  Array<T>::Array(int n, const T &val) {
    assert(n > 0);
    NumberValues = n;
    Values = new T [n];
    assert(Values);
    for (int I = 0; i < n; ++i) {
        Values[I] = val;
    }
}
```

```
// 구축자는 표준배렬로부터 초기화한다.
template<class T>
  Array<T>::Array(const T A[], int n) {
    assert(n > 0);
    NumberValues = n;
    Values = new T [n];
    assert(Values);
    for (int i = 0; i < n; ++i) {
        Values[I] = A[I];
    }
}
```

깊은 복사에 의하여 원천과 대상목록은 깊으면서도 다르게 표시된다. Values자료성원은 같은 값을 가지는 서로 다른 기억기위치를 지정한다. 얕은 복사에서 2개의 목록은 하나의 식을 공유하며 Values자료성원은 같은 기억기위치를 지정한다.

const와 비상수형Array<T>객체를 위한 첨수연산자의 두 다중정의를 목록 14-4에 보여 주었다.

목록 14-4. alist.h에 있는 성원함수들과 연산자들

```
// 복사구축자
template<class T>
  Array<T>::Array(const Array<T> &A) {
    NumberValues = A.size();
    Values = new T [A.size()];
    assert(Values);
    for (int i = 0; i < A.size(); ++i)
        Values[i] = A[i];
    }
// 해체자
template<class T>
  Array<T>::~Array() {
    delete[] Values;
}
// 값주기
template<class T>
  Array<T>& Array<T>::operator = (const Array<T> &A) {
    if (this != &A) {
        if (size() != A.size()) {
            delete[] Values;
            NumberValues = A.size();
```

```
            Values = new T [A.size()];
            assert(Values);
        }
        for (int i = 0; i < A.size(); ++i)
            Values[i] = A[i];
    }
    return *this;
}
// 개별적인 요소들의 값을 찾는 연산자
template<class T>
  const T& Array<T>::operator[](int i) const {
    assert((i >= 0) && (i <size()));
    return Values[i];
}
// 개별적인 요소의 촉진자를 찾고 변경하는 연산자
template<class T>
  T& Array<T>::operator[](int i) {
    assert((i > = 0) && (i < size()));
    return Values[i];
}
```

목록 14-5에서 보조삽입연산자는 두번 다중정의된다. 첫번째 정의는 Array<T>객체의 일반적인 본보기정의이다. 정의는 삽입연산자가 목록에서 표시되는 요소의 형을 위하여 정의될것을 요구한다. 본보기는 한쌍의 괄호안에 목록을 추가한다. 목록의 개별적요소는 공백에 의하여 분리된다. 삽입연산자의 두번째 다중정의는 특수하게 Array<char>객체를 위해서이다. 이 삽입정의는 문자렬형태로 목록을 현시하는데 목록의 개별적char형요소는 출력지령안에서 서로 관련되여 있다.

목록 14-5. alist.h에서 array보조연산자

```
// 본보기삽입연산자
template<class T>
  ostream& operator << (ostream &sout, const Array<T> &A){
    sout << "[";
    for (int i = 0; i < A.size(); ++i) {
      sout << A[i] <<"";
    }
    sout << "]";
    return sout;
}
// Array<char>를 위한 삽입연산자
```

```
ostream& operator<<(ostream &sout, const Array<char> &A) {
    for (int i = 0; i < A.size(); ++i) {
        sout << A[i];
    }
    return sout;
}
```

프로그람 14-2는 Array<T>의 기능을 보여 준다. 객체 A와 B의 정의는 이 요소수와 요소들의 초기값을 지정한다. 객체 C의 정의는 초기화를 수행하기 위하여 표준배렬과 요소수를 리용하는 특수한 구축자를 불러 낸다. C++가 문자렬을 마지막요소가 빈 문자인 상수 char형객체의 배렬로 취급하기때문에 이 구축자를 불러 낸다.

검사프로그람에서 다음의 값주기명령문은 B를 A로 복사한다.

```
A=B;
```

```
#include <iostream>
#include <string>
#include "alist.h"
using namespace std;
int main() {
    Array<int> A(5, 0); //A는 0이 5개이다.
    const Array<int> B(8, 1); //B는 1이 8개이다.
    Array<char> C("hello", 5);//C는 h ,e ,l ,l ,o이다.
    cout << "A = " << A << endl;
    cout << "B = " << B << endl;
    cout << "C = " << C << endl;
    A=B;
    A[5] = 3;
    A[B[1]] = 2;
    cout << "A = " << A <<endl;
    cout << "B = " << B << endl;
    cout << "C = " << C << endl;
    return 0;
}
```

프로그람 14-2. Array<T>에 대한 검사프로그람

값주기 A[5]=3;은 성원첨수연산자의 비상수형만을 리용한다. 그러나 값주기 A[B[1]]=2;은 2개의 성원첨수연산자를 리용하는데 B[1]에 대한 참조는 const배렬객체의 첨수연산자를 리용하며 연산의 결과는 다른 첨자연산자의 파라메터로서 리용된다(값 3). 이 프로그람의 결과는 다음과 같다.

```
A=[0 0 0 0 0]
B=[1 1 1 1 1 1 1 1 ]
```

```
C=hello
A=[1 2 1 1 1 3 1 1 ]
B=[1 1 1 1 1 1 1 1 ]
C=hello
```

문 제

1. 다형성함수란 무엇인가?
2. 2개의 파라메터를 받는 본보기함수를 작성하시오. 첫번째 파라메터의 값이 두번째 파라메터의 값보다 작으면 -1, 같으면 0, 크면 1을 되돌린다.
3. 동적할당배렬을 리용하여 실행되는 Stack클라스의 머리부파일이 있다. 이 자료구조체는 position객체를 가지고 있다.

```
class Stack {
    public:
        Stack(int StackSize = 20);
        ~Stack();
        // 촉진자들
        // 탄창에 위치를 대피
        void Push(const Position &p);
        // 탄창에서 요소를 회복
        // 그 요소를 귀환
        Position Pop();
        bool IsEmpty() const;
    private:
        // 탄창에 보관할수 있는 요소의 최고개수
        int MaxStackSize;
        // 현재탄창에 있는 요소의 수
        int CurrentStackSize;
        // 탄창의 꼭대기
        int StackTop;
        Position *Values;
}
```

이 객체를 제외하고 임의의 객체형을 가지는 탄창을 만들어야 한다. Stack본보기클라스를 만들어 이것을 할수 있다.그것을 Tstack라고 한다. 본보기클라스 Tstack을 정의하시오.

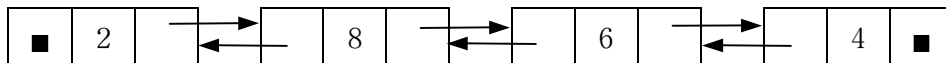
4. Array<T>클라스가 왜 2개의 첨수연산자다중정의를 가지는가? 2개의 다중정의의 리용을 보여 주는 코드를 작성하여 설명하시오.
5. Array에서 최소값을 돌려 주는 서로 다른 성원함수를 추가하여 본보기클라스 Array를 확장하시오. Min()의 실행과 마찬가지로 클라스정의를 변경하시오.

14.5 순차목록

배렬에서 중요한 문제는 크기가 증가, 감소될수 없는것이다.다시 말하면 요소의 수는 콤파일할 때 결정된다. Array<T>객체는 더 좋은 유연성을 가지는데 크기는 실행할 때 결정되며 값주기를 통하여 Array<T>객체는 크기가 다른 목록을 표시할수 있다. push_back()와 resize()와 같은 성원함수를 취급함으로써 조금 더 유연하게 할수 있다. 효과적인 방법은 목록의 임의의 위치의 요소를 삭제하고 삽입하는 SeqList<T>라는 ADT본보기목록을 쓰는것이다. SeqList<T>는 STL용기클라스의 개념에서 list와 류사하다.

ADT가 표시되는 목록에서 순차접근을 제공하기때문에 SeqList<T>라고 이름을 달았다. 이런 목록에서 요소들은 순서대로 배치되며 프로그람이 목록의 개별적요소에 접근할수 있다면 프로그람은 반복자를 계속 전진시켜 목록의 요소에 접근할수 있다. ADT는 앞의 요소에 효과적으로 접근하기 위하여 반복자를 감소시키는 방법을 리용한다. 앞방향, 뒤방향으로 움직이면서 요소들에 접근하는 이런 반복자를 쌍방향반복자(bidirectional iterator)라고 한다.

본보기 SeqList<T> ADT를 실행하기 위하여 2중련결목록(doubly linked list)이라는 자료구조를 리용해야 한다. 2중련결목록은 개별적요소가 3개의 자료성원을 가지는 요소들의 모임이다. 3개 자료성원중에서 하나는 목록값을 나타내며 다른 두 자료성원은 목록에서 자기의 앞뒤요소를 가리키는 지적자이다. 빈 지적자값은 목록의 끝을 의미한다.

다음의 도표는 값이 2, 8, 4, 6인 련결목록(linked list)을 보여 준다.그림에서는 한 요소로부터 다른 요소까지의 관련을 보여 준다. 새까만 4각형은 빈 지적자를 표시한다.

14.5.1 SeqItem클라스본보기

SeqList<T>의 련결목록을 실행하려면 보충적으로 클라스를 정의해야 한다.이 클라스는 관련된 목록의 개별적요소값（즉 요소의 목록값과 지적자）을 표시하는 SeqItem<T>이다. 목록14-6에 SeqItem<T>의 클라스본보기정의를 주었다.

목록 14-6. slist.h에서 클라스본보기 SeqItem

```
template<class T>
  class SeqItem{
    friend class SeqList<T>;
    friend class SeqIterator<T>;
    friend class constSeqIterator<T>;
    protected:
        // 기정구축자
        SeqItem(const T &val)
          :ItemValue(val), Predecessor(0), Successor(0) {
            // 코드가 필요없음
        };
```

```
        private:
            T ItemValue;     // 요소값
            SeqItem *Predecessor;     // 이전 요소의 지적자
            SeqItem *Successor;       // 다음요소의 지적자
};
```

프로그람작성자들이 다른 의미로 SeqList<T>의 요소에 접근하기때문에 이 조건은 사용자가 SeqItem<T>객체를 직접 정의할수 없다는것을 의미한다. SeqItem<T>는 모든 성원함수 **protected**와 모든 자료성원 **private**을 만든다. SeqList<T>객체는 SeqItem<T>객체의 자료성원에 접근할것을 요구한다. 그러므로 SeqItem<T>는 **friend**클라스인 SeqList<T>를 선언한다. 클라스 **friend**는 공개성원이나 비공개성원, 보호성원에는 관계없이 모든 클라스성원에 완전히 접근한다. **friend**함수연산자나 클라스, 변이자 **friend**는 관계가 허가된 함수 연산자나 클라스의 원형으로 적용된다. **friend**의 실지정의는 특수한 문법이 필요없다. 변이자 **friend**는 부록 D에서 구체적으로 설명하였다.

주의

friends를 절대적으로 믿어야 하는가?

friend기구는 어떤 함수나 연산자, 클라스가 기초자료표시를 처리하는것을 조종하지만 정보은폐에 대해서는 중요한 보안구멍을 만들어 놓는다. 그러므로 반드시 필요할 때만 **friend**를 사용해야 한다. 즉 2개의 서로 다른 형의 객체에 직접 접근하는 연산자를 정의해야 할 때만 **friend**를 리용하며 이런 경우에 두 객체에 접근하기 위하여 저준위접근함수를 제공하는것은 실용적이지 못하다. 이때에는 연산자를 두 클라스들의 동료(friend)로 만드는것이 제일 좋은 방법이다.

반복자본보기클라스 SeqIterator<T>도 Seqitem<T>의 동료클라스인데 14.5.8에서 설명한다. 반복자본보기클라스 **const** SeqIterator<T>에 대한 설명은 그만 두기로 하자. 그것의 목적은 **const** SeqList<T>객체에 접근하는 반복자를 제공하는것이다. 2중련결목록의 요소에 적합하므로 SeqItem<T>의 객체 3개의 자료성원을 가진다. 자료성원 ItemValue는 목록값을 표시하며 자료성원 Predecessor와 Succesor는 련결목록의 앞뒤요소를 지적한다.

```
T ItemValue; //요소값
SeqItem *Predecessor;  //앞요소를 가리키는 지적자
SeqItem *Successor ;  //다음요소를 가리키는 지적자
```

SeqItem<T>클라스의 정의는 구축자정의를 포함하고 있다.

```
SeqItem(const t &val)
 :ItemValue(val), Predecessor(0),  Successor(0) {
    //코드는 필요없음
} ;
```

구축자는 파라메터로서 목록요소값 Val을 요구한다. 파라메터 Val은 구축자의 성원초기화목록에서 자료성원 ItemValue를 초기화한다. 2개의 지적자자료성원 Prodecessor와 Successor를 빈 주소로 초기화한다.

SeqList<T>의 클라스본보기정의를 목록 14-7에 주었다. 이 목록은 ADT가 기정구축자, 해체자, 복사구축자, 성원값주기연산자와 같은 표준성원함수를 제공한다는것을 보여 준다. 또한 ADT는 다음의 성원함수를 제공한다.

- size() : 목록의 요소수를 돌려 준다.
- front() const:오른쪽값으로 목록의 첫 요소를 돌려 준다.
- front(): 목록에서 왼쪽값으로 첫 요소를 돌려 준다.

목록 14-7. slist.h에서 클라스본보기 SeqList

```
template<class T>
  class SeqList {
    // 동료클라스들
    friend class SeqIterator<T>;
    friend class constSeqIterator<T>;
    public:
        // typedef정의
        typedef SeqIterator<T>  iterator;
        typedef const SeqIterator<T> const_iterator;
        // 구축자와 해체자
        SeqList();
        -SeqList();
        // 검토자와 접근자
        int size() const;
        T& front();
        const T& front() const;
        T& back();
        const T& back() const;
        // 반복자들
        iterator begin();
        const_iterator begin() const;
        iterator end();
        const_iterator end() const;
        // 보조함수들
        void push_back(const T&val);
        void pop_front();
        // 삽입
        void display(ostream &sout) const
        // 목록함수
        iterator insert(iterator p, const T &val);
        iterator erase(iterator p);
        void clear();
        // 성원값주기와 복사구축자
```

```
        SeqList& operator = (const SeqList &S);
        SeqList(const SeqList<T> &S);
    private:
        // 자료성원
        SeqItem<T> *Front; // 첫 요소의 지적자
        SeqItem<T> *Back; // 마지막요소의 지적자
        int ListLength;  // 요소의 값
};
```

- back() const: 오른쪽값으로 목록의 마지막요소를 돌려 준다.
- back():왼쪽값으로 목록의 마지막요소를 돌려 준다.
- begin()const: 목록의 첫 요소를 지적하는 상수반복자를 돌려 준다.
- begin(): 목록의 첫 요소를 지적하는 반복자를 돌려 준다.
- end()const: 목록에서 마지막요소의 다음에 있는 보호원소를 지적하는 상수반복자를 준다.
- end():목록의 마지막요소다음에 있는 보호원소를 지적하는 반복자를 돌려 준다.
- push_back(const'l'&val):목록의 마지막에서 val의 복사를 추가하는 함수
- pop_front(): 목록의 첫 요소를 삭제하는 함수
- display(Ostream &sout) const: 흐름 sout로 목록을 현시
- insert(iterator p,const T &val):반복자 p가 지적하는 요소의 앞에 val의 복사를 삽입한다. 이 함수는 새 요소를 지적하는 반복자를 돌려 준다.
- clear(): 목록의 모든 요소를 삭제한다.
- erase(iterator p):반복자 p가 지적하는 목록의 요소를 삭제한다. 이 함수는 그전에 삭제된 요소의 다음에 있는 요소를 지적하는 반복자를 돌려 준다.

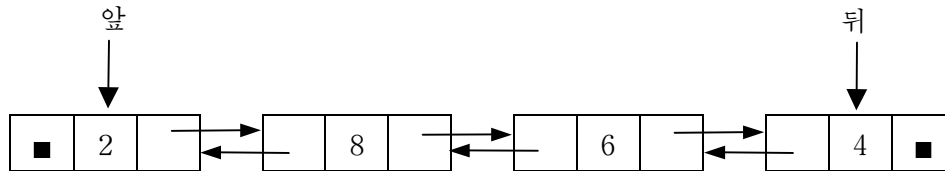
이러한 성원함수들을 제공하기 위하여 SeqList<T>는 3개의 자료성원을 정의한다. int자료성원 ListLength는 목록의 요소수를 나타낸다. front와 back는 목록의 첫 요소와 마지막요소를 표시하는 SeqItem<T>객체를 지적한다. 두개의 반복자클라스 SeqIterator<T>와 const SeqIterator<T>는 SeqList<T>의 동료로 되며 그것들은 SeqList<T>에 접근할수 있다.

14.5.2 SeqList성원함수기초

목록 14-8부터 14-12까지에서 SeqList<T>성원함수의 실현부를 보여 주었다. 목록 14-8은 기정구축자의 정의를 지적한다. 이 구축자는 자료성원의 초기값을 지적하는 초기화목록을 리용한다. 지적자 front와 back는 둘 다 빈 주소로 초기화되며 따라서 빈 목록으로 된다. 자료성원 ListLength는 이 목록상태를 반영하기 위하여 0으로 초기화된다.

SeqList<T>해체자는 자기 기능을 수행하기 위하여 성원함수 clear()를 리용한다. 14.5.6에서 clear()에 대하여 구체적으로 서술하였다. 검토자 size()는 ListLength의 값을 돌려 주는 기능을 수행한다.

front()성원함수의 목적은 둘 다 참조되돌이를 통하여 목록의 첫 요소에 대한 접근을 제공하는것이다. front()함수는 호출되는 함수에 의하여 첫 요소가 조작되는 방법에서 차이난다. 수식자 const를 가진 front()함수는 목록의 첫 요소만 검사하며 const수식자가 없는 front()함수는 목록의 첫 요소를 변화시키기도 하고 검사도 한다.

목록이 비지 않았다면 front()함수는 자기의 동작을 수행할수 있다. 함수는 assert ( ) 명령문으로 이것을 수행한다. 첫 요소가 주어 졌다면 front()가 지적하는 SeqItem<T>객체의 자료성원 ItemValue에 의하여 첫 요소의 값은 유지된다. 따라서 첫 요소가 있다면 front->ItemValue는 그 요소의 값을 가리킨다. front, back와 관련된 목록이 아래의 그림과 같이 표시된다면 front()함수는 값 2를 가진 위치에 대한 참조를 돌려 준다.

목록 14-8. slist.h에서 SeqList성원함수

```
// SeqList기정구축자
template<class T>
  SeqList<T>::SeqList()
    :Listlength(0), Front(0), Back(0) { //
}
// SeqList해체자
template<class T>
  SeqList<T>::~SeqList() {
    clear();
}
// size()목록의 값을 돌려 준다.
template<class T>
  int SeqList<T>::size() const {
    return ListLength;
}
// front(): 목록에서 첫번째 요소에 대한 참조를 돌려 준다.
template<class T>
  T& SeqList<T> ::front() {
    assert (size() !=0);
    return Front -> ItemValue;
}
// front(): 목록에서 마지막요소에 대한 상수참조를 돌려 준다.
template<class T>
  T& SeqList<T>::front() const {
    assert(size() !=0);
    return Front ->ItemValue;
}
```

```
// back(): 목록에서 마지막요소에 대한 참조를 돌린다.
template<class T>
  const T& SeqList<T>::back() {
    assert(size() !=0);
    return Front ->ItemValue;
}
// back():목록에서 마지막요소에 대한 상수참조를 돌려 준다.
template<class T>
  const T& SeqList<T>::back() const {
    assert(size() !=0);
    return Front ->ItemValue;
}
// begin(): 첫 요소를 지적하는 반복자를 창조한다.
template<class T>
  SeqIterator<T> SeqList<T>::begin() {
    return SeqIterator<T> (this, Front):
}
// end(): 마지막요소를 지적하는 반복자를 창조한다.
template<class t>
  SeqIterator<T> SeqList<T>::end() {
    return SeqIterator<T>(this, 0);
}
```

2개의 SeqList<T>back()성원함수는 목록에서 자기의 되돌이값으로서 마지막요소에 접근한다. 이 함수는 마지막요소가 불러 내는 함수에 의하여 처리되는 방법에서만 차이난다. 수식자 const를 가진 back()함수는 목록의 마지막요소만을 검사한다. 수식자 const가 붙지 않은 back()함수는 목록의 마지막요소를 변경도 하고 검사도 한다. 마지막요소가 있다면 그 요소의 값은 back가 가리키는 SeqItem<T> 객체의 자료성원 ItemValue에 의하여 유지된다. 마지막요소가 있다면 Back->ItemValue는 그 요소이다. 우의 그림에서 back()함수는 값 4를 가진 위치에 대한 참조를 돌려 준다.

목록 14-8에서 정의된 다음의 두 성원은 목록처리를 위한 반복자들을 만들어 준다. begin()함수는 목록의 첫 요소를 지적하는 반복자들을 돌려 준다. 반복자는 SeqIterator<T>객체의 구축에 의하여 만들어 진다. 되돌림값을 만들어 주는 SeqIterator<T>구축자는 2개의 파라메터를 가진다. 한 파라메터는 반복자와 관계되는 목록에 대한 지적자이고 다른 파라메터는 목록안의 개별적위치에 대한 지적자이다. SeqIterator<T>구축자는 SeqIterator<T>클라스의 보호성원이다. SeqList<T>성원함수는 SeqIterator<T>의 동료클라스이므로 구축자를 리용할수 있다.

end()함수는 목록에서 마지막요소다음의 보호원소를 가리키는 반복자를 돌려 준다. SeqIterator<T> 클라스는 목록예비의 값을 가리키기 위하여 빈 주소를 리용한다. 이와 같이 SeqIterator<T>(this,0)으로부터 구축된 객체는 현재목록에서 보호원소를 가리키는 반복자를 나타낸다.

목록 14-9의 push_back()함수와 pop_front()는 편리한 함수이다. 목록의 끝에 마디를 추가하고 목

록의 앞에서 마디를 삭제하는 기능이 필요하기때문에 이러한 함수가 리용된다. push_back()함수는 파라메터로서 val와 end()를 가진 성원함수 insert()를 호출하여 목록끝에 val의 값을 삽입할수 있다.

```
insert(end() , val);
```

pop_front()함수는 begin()함수를 파라메터로 하는 erase()함수를 호출하여 자기기능을 수행한다.

```
erase(begin() );
```

insert()함수와 erase()함수는 14.5.4에서 서술하였다.

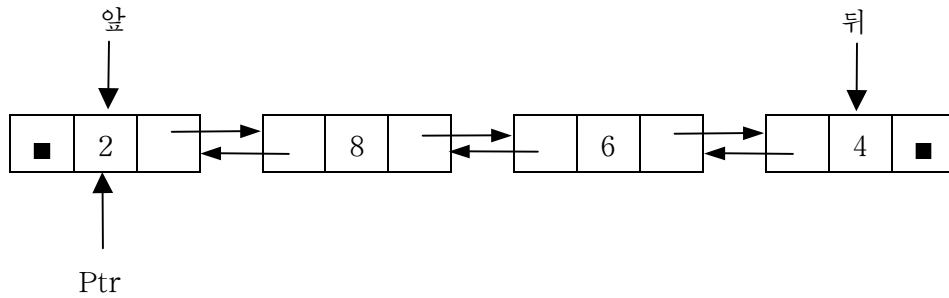
목록 14-9. slist.h안에 있는 SeqList성원함수

```
// 목록의 마지막에 값을 추가
template<class T>
  void SeqList<T>::push_back(const T &val) {
    insert(end(), val);
}
// pop_front():목록의 제일 앞의 값을 지우기
template<class T>
  void seqList<T>::pop_front() {
    erase(begin());
}
// display(): 목록현시
template<class T>
  void SeqList<T>::display(ostream &sout ) const {
    sout << "[" ;
    for (SeqItem<T> *Ptr = Front; Ptr; Ptr = Ptr->Successor)
      sout << " " << Ptr-> ItemValue;
    sout << "]";
}
```

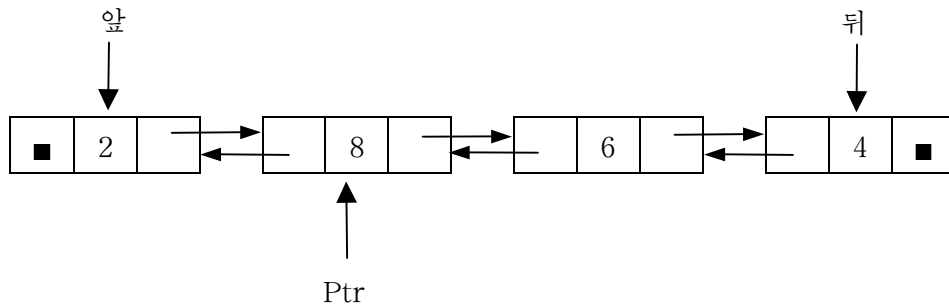
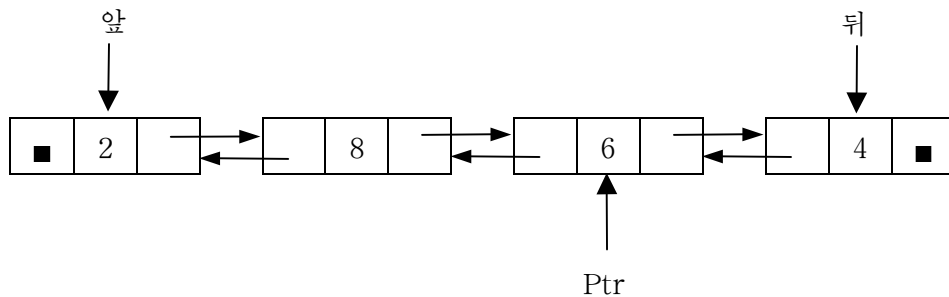
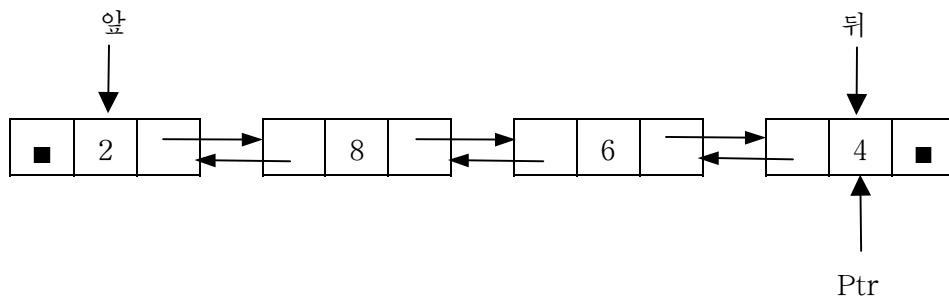
14.5.3 성원함수 display()

목록 14-9의 display()성원함수는 삽입연산자의 다중정의를 쉽게 한다. 목록은 괄호로 둘러 막혀 현시된다. for순환은 목록요소를 현시하게 한다.

```
for (SeqItem<T> *Ptr = Front; Ptr; Ptr= Ptr->Successor)
{
    sout <<" " <<Ptr->ItemValue;
}
```

for순환초기화명령은 초기값이 front인 SeqItem<T>의 지적자 Ptr를 정의한다. 이때 관련되는 객체는 다음과 같이 표시된다.

for순환은 Ptr의 값이 null이 아닌 동안 반복수행한다. for순환안의 실행코드는 한개 삽입명령문으로 구성되였다.

```
sout << " " << Ptr -> ItemValue ;
```

Ptr가 지정하는곳의 SeqItem<T>의 ItemValue(2)는 출력흐름에 삽입된다. 그다음에 for순환이 한번 증가하므로 Ptr는 갱신된다. Ptr의 새값은 관련된 목록의 다음요소의 주소로 된다. 이때 객체는 다음과 같이 표시된다.

Ptr가 null이 아니므로 순환이 다시 반복되며 출력흐름에 8이 삽입된다. Ptr는 다시 갱신된다.이때 객체는 다음과 같이 표시된다.

Ptr가 여전히 null이 아니므로 순환은 반복되며 그리하여 출력흐름에 6이 삽입된다. 이때 객체는 다음과 같이 표시된다.

Ptr는 현재련결목록의 마지막요소값을 가리킨다. 순환은 마지막으로 한번 수행한다. 그리하여 출력흐름에 4가 삽입된다. for순환이 한번 진행되면 Ptr는 null값을 지정한다. 이때 for순환의 검사식은 0으로 판정되므로 순환은 끝난다. 이때 오른쪽닫기괄호가 삽입된다.

```
sout << " ]";
```

련습에서는 목록에서 반복자를 증가시켜 display ( ) 함수의 변화실행을 고찰했다. 목록에 요소를 추가하고 삭제하기 위하여 SeqList<T>insert()와 erase()성원함수를 정의한다.

14.5.4 요소의 추가

목록 14-10에서 insert()함수는 파라메터 p와 val을 가지고 있다. 파라메터 p는 새 요소다음에 있게 될 요소를 지적하는 반복자이다. 새 요소의 값은 val이다. 함수는 p가 목록과 관련되였다는것을 고려하여 시작된다.

```
assert(p. ThisList = = this);
```

새로운 SeqItem<T>객체는 그다음 빈 기억기로부터 요구된다. curr지적자는 이 객체를 가리킨다.선언은 새로운 요구가 제기되였다는것을 보여 준다.

```
SeqItem<T>  *curr = new SeqItem<T>(val);
assert(curr);
```

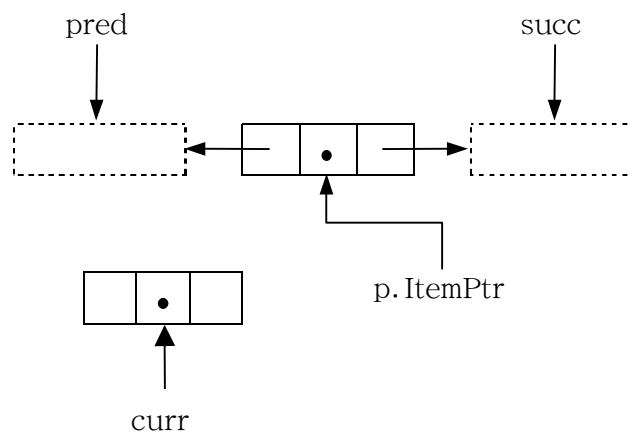
새 요소가 목록에 추가되는 방법은 그것이 목록의 마지막요소로 되는가에 따라 달라 진다. 새 요소가 마지막요소로 되지 않는다면 새 요소의 앞뒤의 차를 표시하는 succ와 pred지적자를 정의할수 있다 (Pred의 값은 p가 현재 가리키는 요소가 목록에서 첫 요소로 된다면 null로 된다).

```
SeqItem<T> *succ = p.ItemPtr;
SeqItem<T> *pred = succ->Predecessor;
```

목록 14-10. slist.h에서 SeqList성원함수

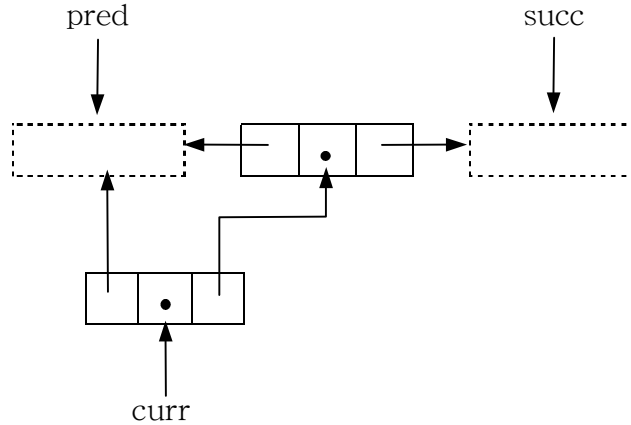
```
// insert():목록에 항목을 추가한다.
template<class T>
  SeqIterator<T> SeqList<T>::
    insert(SeqIterator<T> p, const T &val) {
    // 반복자가 목록을 포함하는가를 확인
    assert(p. ThisList = =  this);
    // 새 요소를 표현하기 위한 목록창조
    SeqItem<T> *curr = new SeqItem<T>(val):
    assert(curr);
    if (p.ItemPtr ! = 0)  {
        // 새 항목의 선두와 마지막을 결정
        SeqItem<T> *succ = p.ItemPtr;
        SeqItem<T> *pred = succ-> Predecessor ;
        // 목록의 변화를 반영하는 관련을 재설정
```

```
        curr-> Successor = succ;
        curr-> Predecessor = pred;
        succ-> Predecessor = curr;
        if (succ = = Front) {
            Front = curr;
        }
        else {
            pred->Successor = curr;
        }
    }
    else { // 반복자 p는 목록요소를 지적하지 않는다.
        if (size() = = 0)  {   // curr는 이 점에서 목록이다.
            Front = Back = curr;
        }
        else {  // curr는 목록의 끝으로 된다.
            curr -> Predecessor = Back;
            Back -> Successor = curr;
            Back = curr;
        }
    }
    // 항목이 추가된다는것을 보여 준다.
    ++ListLength;
    // 새 항목에 반복자를 돌려 준다.
    return SeqIterator<T> (this, curr);
}
```

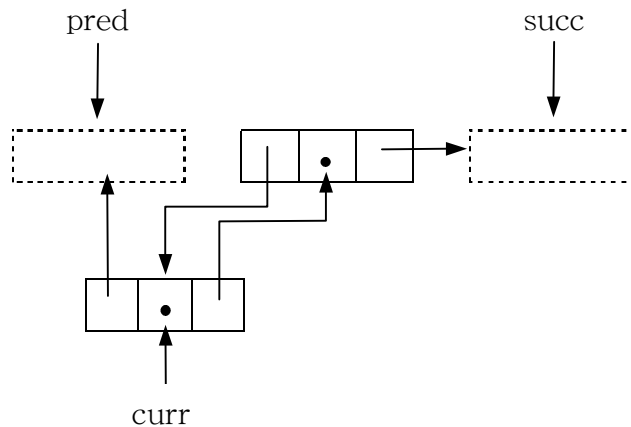
아래의 그림에 이런 경우를 보여 준다. 이 그림과 다음그림에서 점선으로 그려 진 4각형은 관련고리가 null이라는것을 보여 준다. 새까만 원은 요소의 값을 의미한다.

succ지적자와 pred지적자는 새 요소의 후행자(successor)와 선행자(predecessor)에 대한 자료성원

을 정확히 설정하기 위하여 리용된다.

```
curr -> Successor = succ;
curr ->Predecessor = pred;
```

아래에 흥미 있는 객체값주기를 보여 준다.

이 경우에 어떤 요소가 새로운 요소를 뒤따르므로 그 원소의 선행자(succ->Predecessor)를 새 요소로 설정해야 한다.

```
succ->Predecessor = curr;
```

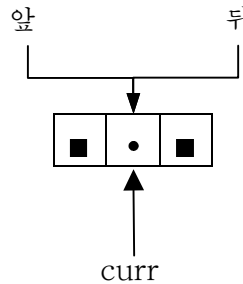
이 값주기에 의하여 객체는 다음과 같이 된다.

새 요소가 목록의 제일 앞에 있는가 없는가에 따라 Front가 그 새 요소를 지적하게 하든가 아니면 후행자(그 새 요소의)의 이전의 선행자(pred - >Successor)가 그 새 요소를 지적하게 해야 한다.

```
if (succ = = Front) {
    Front = curr;
}
else {
    pred->Successor = curr;
}
```

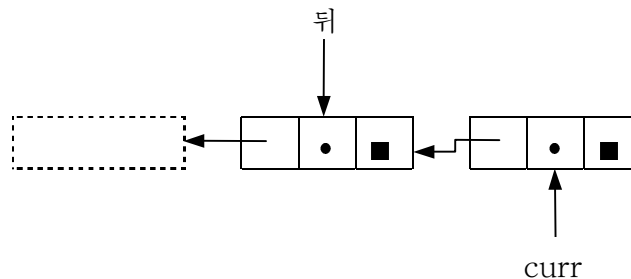
새 요소가 목록의 마지막요소로 되는 경우를 고찰해 보시오. 이 동작은 두개의 보조부분으로 분할된다. 이 부분들은 목록이 비였는가에 관계된다(즉 size()가 값 0을 가질 때).

목록이 현재 비였다면 새로운 요소는 목록의 첫 요소로 되고 마지막요소로도 된다. 그러므로 자료성원 Front와 back는 새 요소를 지적하게 된다.

Front= Back=curr;

이 경우 그것은 새 요소의 구축시 이 성원에 대하여 주어 진 값이므로 새 요소의 후행자와 선행자자료성원이 빈 주소로 설정할 필요는 없다.이러한 값주기명령이 수행된 다음 객체의 모형은 다음과 같다.

만일 목록이 비여 있지 않다면 삽입되는 새 요소의 선행자는 목록의 제일 마지막에 있는 요소를 지적한다.

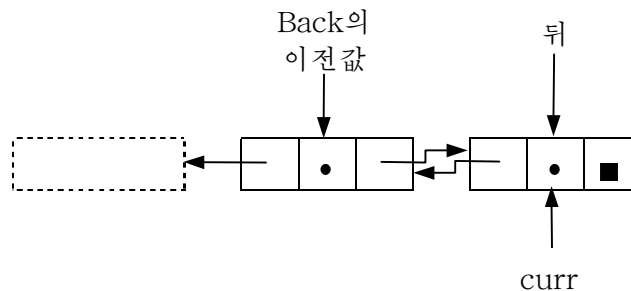
curr-> Predecessor = Back;

이 값주기명령이 수행되면 다음과 같은 모형으로 된다.

이 경우에 새 요소의 선행자가 있어야 한다.

Back-> Successor = curr;

그리고 새 요소가 현재목록의 마지막이라는것을 반영하기 위하여 Back를 변경해야 한다.

Back= curr;

이 값주기명령이 실행되면 객체모형은 다음과 같다.

새 요소를 삽입하는 방법에서 특별히 제기되는 문제가 없으면 목록에 다른 요소가 있는가, 없는가를 나타내는 자료성원 ListLength를 변경해야 한다.

++ListLength;

함수의 되돌림값은 새 요소를 지적하는 반복자이다.

```
return SeqIterator<T>(this, curr);
```

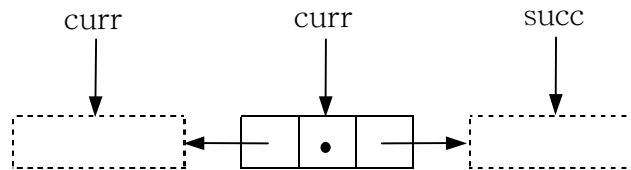
14.5.5 요소의 삭제

목록 14-11의 erase()함수는 2개의 선언으로 시작된다. 첫 선언은 반복자가 현재목록을 지적하는가를 검사한다. 두번째 선언은 삭제할 요소가 있는가를 검사한다.

```
assert(p.ThisList == this);
assert(p.ItemPtr);
```

그다음 erase()함수는 curr, pred와 succ객체를 정의한다. curr객체는 삭제되게 된 객체의 지적자이다. pred객체와 succ객체는 삭제되는 요소의 선행자와 후행자자료성원이 다같이 지적하는 위치에 대한 지적자이다.

```
SeqItem<T> *curr = p.ItemPtr;
SeqItem<T> *pred = curr->Predecessor;
SeqItem<T> *succ= curr->Successor;
```

이러한 코드가 실행되면 객체의 모형은 다음과 같이 된다.

curr가 목록의 첫 요소 혹은 마지막요소를 지정하는가를 결정하기 위하여 검사를 진행해야 한다. curr가 목록의 첫 요소를 지적한다면 삭제되는 요소의 뒤에 있는 요소가 목록의 첫 시작요소로 된다.

```
Front = succ;
```

목록 14-11. slist.h에서 성원함수 SeqList

```
// 목록에서 한개 항목을 삭제
template<class T>
  SeqIterator<T> SeqList<T>::erase(SeqIterator<T> p) {
    // 반복자가 목록을 포함하는가를 확인한다.
    assert(p.ThisList = = this);
    // 반복자가 목록을 지적하는가를 확인한다.
    assert(p. ItemPtr);
    // 목록과 그것의 이웃성원들을 식별한다.
    SeqItem<T> *curr = p.ItemPtr;
    SeqItem<T> *pred = curr->Predecessor;
    SeqItem<T> *succ= curr->Successor;
      // 항목의 제일 앞의 요소인가를 결정한다.
```

```
        if (curr = = Front) {
            // 꼬리지적자는 목록의 머리로 된다.
            Front = succ;
        }
        else {
            // 선행자는 자기의 후행자를 지적한다.
            pred->Successor = succ;
        }
        // 항목이 목록의 끝인가를 결정한다.
        if (curr = = Back) {
            // 선행자는 목록의 끝을 가리킨다.
            Back = pred; }
        else {
            // 후행자는 자기의 선행자를 지적한다.
            succ->Predecessor = pred;
        }
        // 항목을 삭제할수 없다.
        delete curr;
        // 항목이 삭제되였는가를 보여 준다.
        --ListLength;
        // 후행자에 반복자를 돌려 준다.
        return SeqIterator<T> (this, succ);
    }
```

curr가 목록의 첫 요소를 지적하지 않는다면 그 요소의 선행자는 요소의 후행자를 지적하게 된다.

```
pred->Successor = succ;
```

curr가 목록의 마지막요소를 지적한다면 그 요소의 선행자는 목록의 끝으로 된다.

```
Back= pred;
```

curr가 마지막요소를 지적하지 않는다면 이때 요소의 후행자는 요소의 선행자를 지적한다.

```
succ->Predecessor = pred;
```

이렇게 련결이 설정되여 요소가 목록에서 삭제되면 그의 기억기가 비게 된다. 한개 요소가 삭제되였다는것을 나타내기 위하여 ListLength를 변화시킨다.

```
delete curr;
--ListLength;
```

함수를 완성하기 위하여 삭제된 요소의 다음요소에 대한 반복자를 돌려 준다.

```
return SeqIterator<T>(this, succ);
```

14.5.6 모든 요소의 삭제

목록 14-12의 성원함수 clear()는 고려할 부분이 없으므로 erase()보다 더 높은 기능을 수행한다.이 함수는 front의 값에 객체 Ptr를 설정하는것으로 시작된다. Ptr는 련결목록에서 요소들을 순환해 보기 위하여 리용된다.

```
SeqItem<T> *Ptr = Front;
```

순환고리는 Ptr가 목록에서 실제요소를 지정하는 동안 반복된다. 순환고리내에서 림시지적자객체 next는 련결목록의 다음요소(Ptr->successor)를 기억하고 있다. 현재 련결된 목록항목이 일단 삭제만 된다면 Ptr는 순환을 할수 있도록 Next로 재설정된다.

```
while (Ptr) {
    SeqItem<T> *Next = Ptr ->Successor;
    delete Ptr;
    ptr = Next;
}
```

순환이 완료되면 clear()를 끝내기 위하여 SeqList<T>객체의 세 자료성원들이 빈 목록값을 반영하도록 재설정된다.

```
Front = Back = 0;
ListLength = 0;
```

이 련습에서는 반복자를 사용하여 clear()함수의 변화동작을 고찰한다.

목록 14-12. slist.h에서 SeqList clear()성원함수

```
template<class T>
  void SeqList<T>::clear() {
    SeqItem <T> *Ptr = Front;
    while ( Ptr ) {
        SeqItem<T> *Next = Ptr->Successor;
        delete Ptr;
        Ptr = Next;
    }
    Front = Back = 0;
    ListLength = 0;
}
```

14.5.7 SeqList본보기클라스를 리용하는 작은 프로그람

프로그람 14-3은 SeqList<T>본보기클라스의 특성을 보여 준다. 이 프로그람은 옹근수의 목록에 대한 삽입연산과 삭제연산을 진행한다. 이 프로그람에서 삽입연산자의 다중정의가 정확히 되였으며 이제는 그만 보기로 하자. 프로그람 14-3의 출력을 아래에 보여 준다.

```
List: [ ]
Adding: 1
List: [ ]
Adding: 2
List: [1 2]
Adding: 3
List: [1 2 3]
Adding: 4
List:[1 2 3 4]
Adding: 5
List:[1 2 3 4 5]
Removing: 1
Removing: 2
List: [3 4 5]
Removing: all
List: [ ]
Adding: 62154
List: [ 62154 ]
```

```
// 프로그램 14-3. SeqList<T>의 능력을 보여 준다.
#include<iostream>
#include<string>
using namespace std;
#include "slist.h"
int main() {
    SeqList<int> A;
    cout << "List:" << A << endl;
    for (int i=1; i<=5 ; ++i)  {
        cout << "Adding: " << i << endl;;
        A.push_back(i);
        cout << "List:" << A << endl;
    }
    cout << "Removing:" << A.front() << endl;
    A.pop_front();
    cout << "Removing:" << A.front() <<endl;
    A.pop_front();
    cout << "List:" << A << endl;
    cout << "Removing all" <<endl;
    A.clear();
```

```
    cout <<"List:" <<A <<endl;
    cout <<"Adding: " <<62154 <<endl;
    A.push_back(62154);
    cout << "List:" <<A<<endl;
    return 0;
}
```

프로그람 14-3. SeqList<T>본보기클라스의 최종검사프로그람

14.5.8 SeqList반복자클라스

프로그람 14-3에 있는 목록처리연산은 목록요소를 개별적으로 참조한다. 이러한 기능은 SeqList<T>로 동작하는 본보기클라스 SeqIterator<T>에 의하여 제공된다. 일반적으로 반복자클라스는 련관된 용기클라스의 요소에 접근하기 위한 개별적인 수단을 제공한다. 련관된 클라스에 따라 반복자클라스는 참조된 요소를 변경시키기 위한 수단을 제공한다. 본보기클라스 SeqIterator<T> 는 목록의 요소에 접근하기도 하고 변경도 할수 있다. SeqIterator<T>클라스본보기의 정의를 목록 14-13에 주었다.

목록 14-13. slist.h에서 SeqIterator클라스본보기정의

```
template<class T>
  class SeqIterator {
    friend class SeqList<T>;
    public:
        // 기정구축자
        SeqIterator();
        // 특정한 구축자
        SeqIterator(SeqList<T> &L);
        // 목록의 다음요소를 변화
        SeqIterator<T>& operator++();
        SeqIterator<T> operator++(int);
        // 목록의 이전 요소를 변화
        SeqIterator<T>& operator--();
        SeqIterator<T> operator--(int);
        // 상수반복자에 대한 검토자
        const T& operator*() const;
        // 비상수반복자에 대한 검토자 및 변이자
        T& operator*();
        // 같은가를 검사한다.
        bool operator = = (const SeqIterator<T> &p) const;
        // 다른가를 검사한다.
        bool operator! = (const SeqIterator<T> &p) const;
        operator bool() const;
```

```
    protected:
        // SeqList의 지정한 구축자를 리용
        SeqIterator(SeqList<T> *p, SeqItem<T> *q);
    private:
        SeqList<T> *ThisList;  // 반복된 목록의 지적자
        SeqItem<T> *ItemPtr;  // 현재요소의 지적자
};
```

SeqIterator<T>클라스의 기정구축자는 빈 목록과 반복자를 련관시킨다. 다른 공개구축자는 파라메터 L에 의하여 표시되는 목록과 반복자를 련관시킨다. 구체적으로 목록이 비지 않았다면 반복자는 L에서 첫 요소와 련관된다. 그렇지 않으면 반복자가 목록의 보호요소와 련관된다.

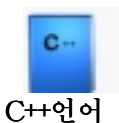
SeqIterator<T>클라스의 보호구축자는 2개의 파라메터를 가지는데 하나는 p이고 다른 하나는 q이다. 파라메터 p는 반복자가 련관되는 목록에 대한 지적자이다. 파라메터 q는 목록에서 개별적인 위치에 대한 지적자이다. 이 구축자는 개별적인 요소의 반복자를 구축하기 위하여 SeqItem<T>에 의해 리용된다.

다음 코드토막의 마지막에서 반복자 a와 b, c가 정의된다. a의 정의는 기정구축자를 리용한다. 그러므로 a는 그 어떤 개별적인 목록요소와 관계되지 않는다. b는 SeqList<**char**>A의 첫 요소와 b를 련관시킨 다른 클라스가 정의한 공개구축자를 리용한다. c정의는 a와 같은 요소와 c를 련관시킨 콤파일러가 제공하는 복사구축자를 리용한다.

```
SeqList<char> A;
A.push_back('x');
A.push_back('y');
A.push_back('z');
SeqIterator<char> a;
SeqIterator<char> b(A);
SeqIterator<char> c=A.begin();
```

SeqIterator<T>반복자클라스는 풍부한 기능을 제공한다. 이 성원의 일부 기능들은 연산자 ++, --, *를 다중정의하여 제공한다. 감소연산자는 련습에서 보기로 하자.

증가연산자는 두번 다중정의되였다. 이러한 다중정의는 연산자가 앞붙이형태와 뒤붙이형태로 리용되게 한다. 어떤 정의가 어떤것인가를 식별하기 위하여 C++는 정의를 뒤붙이형태로 빈 파라메터목록과 련관시킨다. 뒤붙이형태는 **int**파라메터로 된 정의와 관련된다.

C++언어

앞붙이와 뒤붙이

증가 및 감소연산자는 뒤붙이 또는 앞붙이로 될수 있다. ++와 --연산자에서 앞붙이와 뒤붙이 다중정의를 구별하기 위하여 **int**인수는 함수가 연산자의 뒤붙이응용에 리용된다는것을 지적하는데 쓰인다. 이 인수가 리용되지 않으면 연산자의 다중정의중에서 앞붙이와 뒤붙이선언사이를 콤파일러가 구별하도록 하는것은 순수한 문장구성문제이다. 다중정의가 앞붙이인가 뒤붙이인가를 기억하는 방법은 앞붙이가 다른 앞붙이연산자(*, ! 등)와 같은 인수를 전혀 가지지 않으며 모조인수는 《짝수》뒤붙이 ++와 --에 대해서만 사용된다는것이다.

만일 반복자가 목록에서 마지막요소와 련관되여 있다면 ++연산자는 목록에서 다음요소와 반복자를 련관시킨다. 만일 반복자가 마지막요소와 련관된다면 ++연산자는 보호원소와 반복자를 련관시킨다. 앞붙이는 갱신된 후에 반복자의 되돌림값을 참조한다. 뒤붙이는 갱신되기전에 반복자의 되돌림값을 가진다.

참조해제연산자 *도 두번 다중정의된다. 이것은 둘 다 반복자에 련관되는 현재요소에 접근한다. 성원함수변경자 **const**를 가진 다중정의는 오른쪽값호출이 **const** SeqIterator <T>객체에 의하여 요소에 접근한다. 성원함수변경자 **const**를 가지지 않은 다중정의는 SeqIterator<T>객체에 의하여 요소에 대한 왼쪽값접근을 하게 한다.

다음의 코드는 증가 및 참조해제연산자를 리용하여 정의된 목록 a의 처음 두 요소를 현시한다.

```
cout << *b <<end1;
++b;
cout << *b<<end1;
```

이 코드의 출력은 다음과 같다.

```
x
y
```

비상수반복자의 참조해제연산자가 왼쪽값을 돌려 주기때문에 *b에 대한 값주기로서 련관되는목록을 변경시킬수 있다.

```
*b = 'w'
cout << "List: " <<A << end1;
```

출력은 다음과 같다.

```
List: [x w z]
```

같기연산자 ==와 안같기연산자 !=는 반복자객체를 위하여 다중정의된다. 같기연산자는 반복자가 같은 요소와 련관되였다면 참을 되돌리고 그렇지 않으면 거짓을 돌려 준다. 안같기연산자는 이와 반대동작을 수행한다.

성원연산자 **bool**이 정의되였다. 이 성원은 반복자가 목록에서 실제요소와 련관되여 있다면 참을 돌려 준다. 또한 반복자가 목록에서 임의의 요소와 련관되여 있지 않다면 거짓값을 돌려 준다. 이 성원이 존재하므로 반복자가 론리표시를 할수 있는 기능을 제공해 준다. 실례로 자동변환은 다음의 코드토막의 **for**명령에 대한 검사프로그람에서 보여 준다.

```
for (SeqIterator<char> p=A.begin(); p; ++p)  {
    cout << "Element; " <<*p;
    if (p == A.begin()) {
        cout << "front";
    }
    cout << end1;
}
```

for검사식은 성과적인 콤파일을 위하여 론리값으로 평가되여야 한다. 검사식이 론리식이 아니라면 콤파일러는 **bool**변환을 적용한다. 반복자가 목록요소를 지적할 때 **bool**변환은 SeqIterator<T>객체가 참

으로 정의하였으므로 p를 참으로 평가되고 for순환본체가 실행된다.

p가 목록에서 첫 요소를 지적하기때문에 순환의 첫 반복자는 다음과 같은 출력을 만든다.

Element:xfront

순환의 마지막식 ++p는 반복자가 요소값 "w"를 지적하게 한다. 순환이 다시 반복되면 출력은 다음과 같다.

Element: w

이러한 식의 평가는 반복자 p가 "z"를 표시하는 요소를 지적하게 한다. 순환이 또다시 진행되면 출력은 다음과 같다.

Element:z

마지막식의 평가는 반복자를 보호원소와 련관시킨다. 검사식은 이때 평가되며 우의 동작에 기초하여 변환을 하며 검사식은 거짓으로 평가된다.

14.5.9 SeqIterator반복자클라스의 실현부

SeqIterator<T>성원함수와 연산자의 실행은 목록 14-14와 목록 14-15에 보여준다. 이러한 성원들의 모든 정의는 서로 관계를 가지고 있다.

SeqIterator<T>기정구축자는 빈 목록으로 된 반복자와 련관되여있다. 그래서 자료성원 ThisList와 ItemPtr가 초기화목록을 리용하여 빈 주소로 설정된다. 일반적으로 ThisList는 목록지적자를 유지하며 ItemPtr는 목록안에 있는 요소의 위치를 지적한다.

다른 SeqIterator<T>공개구축자는 L에 의하여 표시되는 목록으로 구축된 반복자와 련관된다. 자료성원 ThisList의 초기값은 파라메터 L에 표시된 목록의 주소이다. 자료성원 ItemPtr의 초기값은 L.Front이다. 이렇게 L이 텅 빈 값목록이 아니라면 반복자는 L의 첫 요소와 련관되며 그렇지 않으면 반복자는 L의 보호원소요소와 련관된다.

보호된 SeqIterator<T>구축자는 초기화목록을 리용하여 자기의 자료성원을 설정한다.이 구축자와 함께 자료성원 ThisList초기값은 파라메터 p가 가리키는 목록에 대한 지적자이며 ItemPtr의 초기값은 파라메터 q가 가리키는 목록위치에 대한 지적자이다. 구축자정의가 q를 검사하지 않기때문에 목록 L과 련관없는 지적자값을 받을수 있다. 구축자의 안전판번호는 이 련습에서 고찰하게 된다. 이 구축자는 SeqList<T>의 반복자성원함수에서 리용된다.이 성원함수는 SeqList<T>가 SeqIterator<T>에 대한 **friend**클라스이므로 보호구축자를 리용할수 있다.

SeqIterator<T>반복자의 앞붙이증가연산자 ++는 반복자가 Null값이 아닌 첫 번째 값을 처리하기 위하여 동작한다. 반복자가 null값이 아니라면 그것은 식 Itemptr->Successor를 평가하게 한다. 식의 값은 목록에서 다음요소의 주소이거나 빈 주소일수도 있다.

목록 14-14. slist.h에서 SeqIterator성원

```
// SeqIterator():가상구축자
template<class T>
  SeqIterator<T>::SeqIterator():ThisList(0), ItemPtr(0) {
    // 코드필요없음
```

```
}
// SeqIterator(): 특정한 공개구축자
template<class T>
  SeqIterator<T>::SeqIterator(SeqList<T> &L)
    :ThisList(&L), ItemPtr(L.Front) {
    // 코드필요없음
}
// 특정한 보호구축자
template<class T>
  SeqIterator<T>::SeqIterator(SeqList<T> *p, SeqItem<T> *q)
   :ThisList(p), ItemPtr(q) {
    // 코드필요없음
}
// ++: 선행자를 만든다(앞붙이 판번호)
template<class T>
  SeqIterator<T>&SeqIterator<T>
  ::operator++(){
    assert(ItemPtr!=0);
    ItemPtr=ItemPtr->Successor;
    return *this;
}
// ++:선행자를 만든다(뒤붙이판번호)
template<class T>
SeqIterator<T>& SeqIterator<T>
  SeqIterator<T> SeqIterator<T>::operator++(int) {
    assert(ItemPtr !=0);
    SeqIterator<T> remember(*this);
    ItemPtr = ItemPtr->Successor;
    return remember;
}
// ++:상수반복자를 위한 검토자
template<class T>
  const T& SeqIterator<T>::operator*() const {
    assert(ItemPtr !=0);
    return ItemPtr->ItemValue;
}
// *: 빈 상수반복자를 위한 검토자/변이자
template<class T> T& SeqIterator<T>::operator*() {
    assert(ItemPtr !=0);
    return ItemPtr -> ItemValue;
}
```

식 ItemPtr->Successor의 값은 ItemPtr의 정확한 값이다.

```
Itemptr = ItemPtr-> Successor;
```

SeqIterator클라스가 SeqItem<T>의 클라스와 **friend**클라스이기때문에 SeqItem<T>클라스의 비공개 자료성원은 호출될수 있다. 연산을 진행하기 위하여 반복자의 참조를 돌려 준다.

```
return *this;
```

SeqIterator<T>반복자의 뒤붙이증가연산자 ++는 반복자가 빈 값이 아닌 첫 요소를 처리한다. 반복자는 객체에 대한 정의를 하고 객체 remember에 그것을 기억한다. 객체 remember는 연산의 값을 돌려준다.

```
SeqIterator<T>remember(*this);
```

ItemPtr는 다음의 요소를 지적하기 위하여 변경된다.

```
ItemPtr = ItemPtr->Successor;
```

연산을 진행하기 위하여 반복자 remember를 돌려 준다.

```
return remember;
```

증가연산자와 같이 참조해제성원연산자 *도 처음에 결합된 요소들이 있는가를 결정한다. 만일 있다면 두개의 참조해제연산자의 되돌림값은 련관된 요소의 값으로 되며 그것은 간단히 ItemPtr->ItemValue로 표시된다.

2개의 참조해제연산자는 그것들이 오른쪽값 혹은 왼쪽값을 되돌리는가에 따라 차이난다. 같기, 안같기, 삽입연산자와 **bool**은 목록 14-15에 다중정의되여 있다.

목록 14-15. slist.h에서 SeqIterator성원함수와 보조연산자들

```
// ==같은 요소를 참조하는 반복자가 동작
template<class T>
  bool seqIterator<t>::operator = = (const SeqIterator<T> &p)
    const {
    return ItemPtr = = p.Itemptr;
}
// !=: 서로 다른 요소를 참조하는 반복자를 동작
template<class T>
  bool seqIterator<t>::operator != (const SeqIterator<T> &p)
    const {
    return ! (*this = = p);
}
// bool: 이 성원은 반복자가 한개 요소를 지적하는가를 가리킨다.
template<class T>
  bool seqIterator<t>::operator bool () const{
    return *this != this -> ThisList -> end();
```

```
}
// <<: 목록반복자의 삽입연산자를 다중정의
template<class T>
  bool seqIterator<t>::operator << (ostream &sout,
    const SeqIterator<T> &a) {
    return  sout << *a;
}
```

갈기연산자 ==는 추출되는 객체의 ItemPtr자료성원들과 오른쪽연산자 p가 같은 위치를 지적하는가를 검사한다. 지적자값이 갈다면 연산자는 참을 돌려 주며 지적자값이 다르다면 연산자는 거짓을 돌려 준다.

```
return ItemPtr = =p.ItemPtr;
```

안갈기연산자 !=는 호출되는 객체의 자료성원과 오른쪽연산자 p가 다른가를 검사한다. 연산자는 갈기연산자에 의하여 실행될수 있다.

```
return ! (*this == p);
```

일반적으로 다른 클라스연산자의 개념으로부터 클라스연산자의 정의는 직접연산을 실행하는것보다 정의에서 우선도가 더 높다.

bool형변환연산자는 목록 14-5에서 정의하였다. 형변환연산자는 반복자값을 bool형값으로 변환한다. 이 값은 반복자가 ≪보호원소≫를 지시하는가 안하는가를 검사한다. 반복자가 보호원소를 가리키고 있다면 거짓을 되돌리고 그렇지 않다면 참을 돌려 준다. 특히 형변환은 안갈기연산자를 리용하여 실현할수 있다. 연산자는 정의된 반복자와 명백히 보호원소를 지적하고 있는 반복자를 비교한다.

```
return *this != this -> ThisList -> end();
```

봉사받는 측면에서 보조출력연산자는 SeqIterator<T>반복자객체에 대하여 다중정의된다.그 과제는 현재요소와 련관된 값을 그의 출력지령연산수 Sout에 출력하는것이다.

이것으로 순차목록 ADT와 형태적인 다형성에 대한 서술을 끌낸다. 다음절에서는 다형성을 구체적으로 본다.

문 제

6. 증가연산자에 대하여 다중정의된 앞붙이와 뒤붙이를 어떻게 구분하는가?
7. SeqList<T>클라스를 리용하여 파일로부터 문자렬목록을 입력하는 프로그람을 작성하시오. 프로그람은 중복되는 문자렬을 지우면서 문자렬을 쓰기한다.
8. SeqList<T>를 리용하여 화면자리목록을 입력하고 그것을 Position목록에 기억시킨다. 그다음 매 위치에서 작은 CircleShapes를 그리시오.

14.6 다형성

13장에서 설계한 여러가지 도형을 리용하여 그림을 조작하는 의뢰기프로그람을 작성하려고 한다면 사용자는 매 그림객체가 하나의 작은 그림을 구성하는 여러가지 도형들을 목록으로 하는 그림객체들의 집합체를 리용해야 한다.

일반적으로 이러한 목록은 원 , 4각형, 3각형 등을 포함하는 이종목록일것이다. 이종목록을 처리하자면 다형성의 능력이 필요하다. 어떤 자료형의 객체가 전혀 다른 도형의 집합을 표시할수 있는가? 첫째로 Shape가 여러가지 모양을 나타내는 기초클라스이므로 간단히 Shape배렬을 리용하는것이다. 결국 CircleShape도 Shape이며 RectangleShape도 Shape이며 TriangleShape도 Shape이다. 아래에 소개한 S, R, T, C객체는 창문 W와 위치 P가 미리 정의되였다고 하자.

```
SquareShape S(W, P, Blue, 1);
TriangleShape T(W, P, Red, 1);
RectangleShape T(W, P, Yellow, 3, 2);
CircleShape C(W, P, Yellow, 4);
```

또한 Shape객체들의 배렬 A가 다음과 같이 정의되였다고 가정하시오.

```
Shape A[4]={S, T, R, C};  //그림 A는 4개의 모양으로 구성된다.
```

그러면 아래의 순환을 리용하여 그림 A를 그릴수 있다.

```
for (int i = 0; i<4; ++i) {
    A[i].Draw();
}
```

A가 Shape객체목록이고 Shape::Draw()성원함수가 없으므로 그릴수 없다. 다음시도는 목록 14-6에 보여 준것처럼 Shape클라스에 Draw()성원함수를 추가하는것이다. Shape객체가 충분히 규정되지 않으므로 Draw()함수가 표시할수 있는 객체의 특성으로써 객체의 색갈을 지적한다. 그러나 for순환은 여전히 요구대로 움직이지 않는다. 매 순환마다 호출되는것은 Shape::Draw()함수인데 바로 추출하는 객체의 색갈을 표시한다. 특히 S가 SquareShape를 표시한다고 해도 A[0]은 Shape이다. S를 리용하면 A[0]의 초기화에서 모든 SquareShape의 속성을 A[0]에 대입하지 않았으며 달리 Shape성원들만이 복사된다.

목록 14-16. Draw()함수를 가진 Shape정의

```
class Shape:public WindowObject {
    public:
        Shape(SimpleWindow &w, const Position &p,
        const color c = Red);
        color Getcolor() const;
        void SetColor () (const color c);
        void Draw() {   cout << GetColor(); };
    private:
        color Color;
}
```

그 다음시도는 Shape지적자들의 목록 A를 정의하는것이다.

```
Shape *A[4] = { &S, &T, &R, &c};
```

A의 함수가 더 이상 Shape객체가 아니고 Shape객체들에 대한 지적자이므로 앞서 본 for순환이 리용될수 없다. 다음의 for순환을 분석하시오. 이 순환이 요구하는 동작을 수행하는가?

```
for ( int i=0; i<4 ; ++i)
    A[i] -> Draw();
```

대답은 기초클라스 Shape가 어떻게 정의되는가에 관계된다. 기초클라스 Shape가 목록 14-17과 같이 정의된다면 우에 있는 순환이 요구대로 동작하게 된다. 목록 14-17에서 성원함수 Draw()는 그 앞에 virtual이 붙었다.

목록 14-17. 가상Draw()함수를 가진 Shape클라스

```
class Shape:public WindowObject {
    public:
        Shape(SimpleWindow &w, const Position &p,
        const color c = Red);
        color GetColor() const;
        void SetColor(const color c);
        virtual void Draw();//virtual 함수
    private:
        color Color;
}
```

이 순환에서 i가 0일 때 성원함수 SquareShape::Draw()가 호출되며 i가 1일 때는 TriangleShape::Draw()가 호출된다. 이 호출을 결정하는것은 지적자의 형이 아니고 지적자가 귀착된 위치에 있는 객체의 형이다. 동작이 진행되는 리유는 기초클라스 Shape의 Draw()성원함수가 virtual함수이기때문이다. 가상함수는 참조해제된 지적자가 참조객체에 의하여 호출될 때 실지함수가 지적자의 참조를 나타내는 객체의 형에 따라 결정된다. 파생함수의 정의는 기초클라스의 정의를 재정의한다. 일반적으로 어느 가상함수가 리용되는지는 번역시에 결정할수 없으며 반드시 실행시에 결정된다. 결과 비가상함수를 호출할 때보다 가상함수의 호출에 대한 조작시간이 더 크다.

경험

앞으로의 확장을 위한 유연성

가상함수호출에서 어느 함수가 호출되는가를 결정하는것은 실행할 때까지 지연되므로 함수를 제공하는 파생클라스가 실행되지 않거나 정의되지 않아도 그의 본체에서 가상함수를 호출하는 함수를 콤파일할수 있다. 이 능력은 서고를 설계하는 쏘프트웨어제작자들에게 중요하다. 서고를 리용하는 사람은 파생클라스를 작성하고 쏘프트웨어제작자의 원천파일이 없이도 서고함수를 리용할수 있다.

14.7 가상함수의 의미

앞부분의 for순환에서 A[I]->Draw()는 다형성의 리용을 보여 준다. 즉 객체의 형에 따라 서로 다른 동작을 수행한다. 이러한 다형성능력은 A를 여러가지 집합으로 만든다. 가상함수를 포함하는 리유를 보기 위하여 목록 14- 8에 있는 정의가 다음의 각이한 실례와 같다고 가정하시오.

첫 실례와 같이 다음의 코드토막의 결과를 결정하시오.

```
A.Display();
B.Display();
C.Display();
Ptr = &A;
Ptr -> Display();
Ptr = &B;
Ptr -> Display();
Ptr = &C;
Ptr -> Display();
```

첫 3개 행의 결과는

```
BaseClass Display
DerivedClass1 Display
DerivedClass2 Display
```

이다. A, B, C가 각각 BaseClass, DerivedClass1, DerivedClass2이고 따라서 Display()함수들인 BaseClass:Display(), DerivedClass1:Display() 그리고 DerivedClass2:Display()를 호출되기때문에 이러한 결과가 생긴다.

다음 3개 행의 결과는

```
BaseClass Display
DerivedClass1 Display
DerivedClass2 Display
```

이다. 이 결과는 Ptr가 BaseClass의 지적자이기때문에 생긴다.

어느 지적자가상함수가 호출되는가 하는것은 실행시에 결정된다. 다음의 코드를 실행해 보시오.

```
A=B;
A.Display();
A=0;
A.display();
```

결과는

```
BaseClass Display
BaseClass Display
```

이다.

목록 14-18. 가상함수를 리용한 실례

```
class BaseClass {
    public:
        BaseClass() { return; };
        virtual void Display() {
            cout << "BaseClass Display"<< endl;
        };
        void Print() {
            cout << "BaseClass Print "<< endl;
        };
};
class DerivedClass1:public BaseClass {
    public:
        DerivedClass1() { return; };
        virtual void Display() {
              cout << "DerivedClass1 Display" << endl;
        };
        void Print() {
            cout << "DerivedClass1 Print " << endl;
        };
};
class DerivedClass2:public BaseClass {
    public:
        DerivedClass2() { return; };
        virtual void Display() {
            cout << "DerivedClass2 Display"<< endl;
        };
        void Print() {
            cout << "DerivedClass2 Print " << endl;
        };
};
class DerivedClass3:public BaseClass {
    public:
        DerivedClass3() { return; };
        virtual void Display() {
            cout << "DerivedClass3 Display"<< endl;
        };
        void Print() {
            cout << "DerivedClass3 Print " << endl;
```

```
        };
};
void OutPut(BaseClass &R) {
    R.Display();
}
BaseClass A;
DerivedClass1 B;
DerivedClass2 C;
DerivedClass3 D;
BaseClass *Ptr;
```

A가 BaseClass이므로 이런 결과가 생긴다. 따라서 A에로의 값주기는 BaseClass의 다른 성원만을 복사하는데 이것은 호출되는 Display()함수가 바로 BaseClass의 성원함수라는것을 의미한다.

목록 14-18의 Output()함수가 아래와 같이 호출되면

```
Output (A);
Output (B);
Output (C);
```

출력결과는

```
BaseClass Display
DerivedClass1 Display
DerivedClass2 Display
```

이다.

이 결과는 Output()의 파라메터가 참조형파라메터이기때문에 생긴다. 따라서 참조된 객체의 자료형에 따라 어느 가상성원함수가 호출되는가가 결정된다. 목록 14-18에 있는 객체를 리용하는 다음의 코드를 분석하시오.

```
Ptr = &A;
Ptr -> Print();
Ptr = &B;
Ptr -> Print();
Ptr= &C;
Ptr -> Print();
```

이 코드의 출력결과는

```
BaseClass Print
BaseClass Print
BaseClass Print
```

이다. 이 출력결과는 성원함수 Print()가 가상함수가 아니기때문에 생긴다.

따라서 번역할 때 어느 성원함수가 호출되였는가를 결정하는데 우의 코드에서는 차례로 기초클라스 성원함수를 호출한다. 파생된 클라스들인 DerivedClass1와 DerivedClass2의 Display()함수들은 BaseClass의 Display()함수처럼 **virtual**수식어가 명백히 주어 지지 않아도 가상함수이다. 수식자는 파생된 클라스의 성원함수가 기초클라스의 가상함수와 이름이 같고 파라메터가 같으면 자동적으로 가상화되므로 반드시 필요한것은 아니다. 그런 경우에 수식어 **virtual**이 필요하지 않지만 일반적으로 독자들이 리해할수 있도록 수식어를 붙인다.

목록 14-18에서는 BaseClass로부터 DerivedClass3을 파생한다. 이 클라스는 BaseClass의 Display()와 파라메터가 다른 성원함수 Display()를 정의한다. 이때 BaseClass성원함수 Display()가 교갑화되였으므로 다음의 명령문은 옳지 않다.

```
D.Display() // 틀림: Display()는 교갑화되였다.
```

그러나 다음의 코드토막에서 Display()호출은 옳다.

```
D.Display(3);
D.BaseClass::Display(),
Ptr = &D;
Ptr -> Display();
```

출력결과는

```
DerivedClass3 Display 3
BaseClass Display
BaseClass Display
```

이다.

첫번째로 추출된 Display()는 명백히 DerivedClass3의 Display()를 호출한다. 두번째는 령역결과연산자를 리용하여 BaseClass의 Display()를 명백히 호출한다. DerivedClass3객체가 파라메터가 없는 Display()성원함수를 가지지 않으며 DerivedClass3객체가 BaseClass객체이므로 Ptr->Display()는 BaseClass의 Display()를 호출한다.

기초클라스성원함수를 재정의할 때 파생된 클라스함수는 귀환값형만이 다를수 없다. 두 함수의 파라메터도 달라야 한다.

경험

재리용성을 가정한다.

클라스를 정의할 때 그것의 파생된 클라스의 기초클라스로서 결코 쓰일수 없다는것을 모르는 경우에는 클라스에 가상해체자를 정의한다. 실례로 Shape클라스는 함수본체가 빈 해체자를 가질것이다. 해체자를 가상화하면 모든 파생된 파괴자들도 역시 가상화될것이며 객체가 삭제될 때 정확한 해체자가 문맥에 관계없이 호출되게 된다.

14.8 추상기초클라스

가상성원함수가 실현부를 가지지 않으면 순수가상함수(pure virtual function)로 된다. 순수가상함수는 클라스정의에서 그 함수에 null주소를 할당하여 정의한다. 실례로 목록 14-19에 있는 Shape클라스정의에서 그의 성원함수 Draw()는 순수가상함수이다. 순수가상함수를 가진 클라스를 추상기초클라스

(abstract base class)라고 한다.

목록 14-19. 순수가상함수 Draw()를 가진 Shape클라스

```
class Shape:public WindowObject {
    public:
        Shape(SimpleWindow &w, const Position &p,
          const color c = Red);
        color GetColor(0 const;
        void SetColor(const color c);
        virtual void Draw() = 0; //순수가상함수
    private:
        color Color;
};
```

추상기초클라스는 실현부를 가지지 않으므로 구축을 통하여 추상기초클라스형객체를 정의하려는것은 옳지 않다. 특히 귀환값이나 값파라메터로서 추상기초클라스형객체를 리용할수 없다.

따라서 아래에 준 선언과 원형은 맞지 않는다.

```
Shape S; // 틀림: Shape정의가 없다.
Shape f();  // 틀림: 되돌이값으로 리용할수 없다.
void f(Shape s); // 틀림; 값파라메터로 리용할수 없다.
```

구축이 필요없으므로 추상기초클라스를 참조할수 있다. 아래에 Shape를 리용하는 다음의 선언과 원형은 옳다.

```
TriangleShapeT(W, P, Red, 1);
Shape &R = T;  // T는 Shape인 TriangleShape이다.
Shape &F();     // 현재 존재하는 Shape에 대한 참조를 되돌릴수 있다.
void G(Shape &s); // 참조로서 현재 존재하는 Shape를 넘길수 있다.
```

추상기초클라스를 리용하는 기본목적은 파생클라스의 표준대면부를 제공하는데 리용된다. 추상기초클라스는 파생클라스가 지원하여야 할 대면부를 서술한다. 그러나 어떤 때는 추상기초클라스를 그 기능이 파생된 클라스에로 넘긴다. 이런 리유로부터 Draw()는 목록 14-19에서 순수가상함수로 작성되였는데 지적된 모양을 그리는 동작을 파생클라스에 넘긴다.

다음의 실례에서 가상기초클라스 Shape의 기능을 리용하여 간단한 선을 그릴수 있다. 이 동작을 실행하는 2개의 함수를 프로그람 14-4에 주었다. 프로그람결과는 그림 14-1과 같다. 함수이름은 BuildHouse()와 DrawFigure()이다. 두 함수가 다 SeqList<Shape>형순차목록객체를 조작한다. 현재 목록은 직선을 표시하는데 필요한 모양 등을 추가하도록 한다.

```
// 프로그람 14-4 : 집을 그린다.
#include "shape.h"
```

```
#include "triangle.h"
#include "circle.h"
#include "rect.h"
#include "square.h"
#include "slist.h"
// 원형선언
SeqList<Shape*> BuildHouse(SimpleWindow &W);
void DrawFigure(SeqList<Shape*> *F);
// ApiMain():집의 건설과 그리기를 관리한다.
int ApiMain() {
    SimpleWindow Window("House", 10, 10) ;
    Window.Open();
    SeqList<Shape*> *DreamHouse = BuildHouse(Window);
    DrawFigure(DreamHouse);
    return 0;
}
// BuildHouse() : 집을 만드는 기초모양을 리용한다.
SeqList <Shape*>* BuildHouse(SimpleWindow &W) {
    // 집은 여러개의 부분으로 이루어 진다.
    SeqList<Shape*> * House = new SeqList<Shape*>;
    House -> push_back( // 집은 벽체를 가진다.
        new SquareShape(W,Position(5,7), Blue, 5));
    );
    House -> push_back(// 삼각형지붕
        new TriangleShape(W,Position(5,3), Blue, 5));
    );
    House -> push_back(// 해빛창
        new CircleShape(W,Position(5,7.75), Blue, 5));
    );
    House -> push_back(// 문
        new RectangleShape(W,Position(5,8.5), Blue, 5));
    );
    House -> push_back(// 왼쪽창문
        new SquareShape(W,Position(4, 6), Blue, 5));
    );
    House -> push_back(// 오른쪽창문
        new SquareShape(W,Position(6, 6), Blue, 5));
    );
    return House;
```

```
}
// DrawFigure(): F 목록에 있는 모양들을 그린다.
void DrawFigure(SeqList<Shape*>*F) {
    for (SeqIterator<Shape*> P = f -> begin(); P; ++P) {
        (*(*P)).Draw();
    }
}
```

프로그람 14-4. house.cpp로 집을 그리고 현시하기

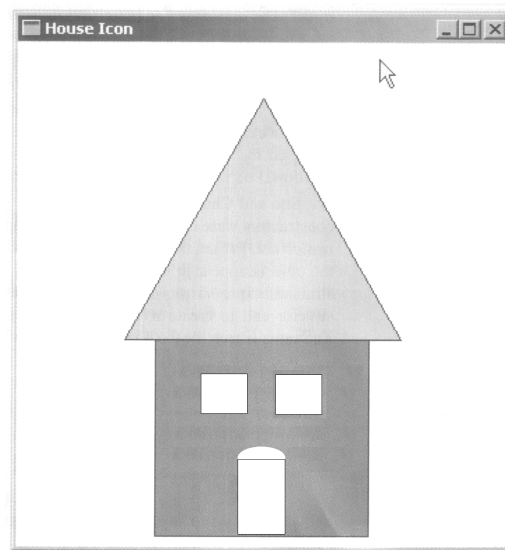



그림 14-1. BuildHouse()와 DrawFigure()를 리용한 집그리기

BuildHouse()함수는 SeqList<Shape*>형련결목록에 대한 지적자를 돌려 준다. 그 목록의 매 요소는 그리기에 필요한 모양에 대한 지적자이다. 그림을 구성하는 모양들과 그것을 지적하는 요소들을 가진 련결목록은 BuildHouse()을 완성한 후에 존재하므로 이 객체들은 BuildHouse()에 의하여 구축된다.

선그리기는 아주 간단하다. 련결목록의 요소들은 6개의 모양만을 지적한다. 그 모양들은 벽과 지붕, 문, 우에 낸 창, 그리고 두개의 창문이다. 벽과 창문은 SquareShape객체이고 지붕은 TriangleShape객체, 문은 Rectangle- Shape객체이며 문우에 난 창문을 먼저 그리고 그다음 문을 그리는 방법으로 구성된다. 문의 웃부분이 문우에 난 창을 가리우므로 문우에 난 창은 반원모양을 가진다. DrawFigure()함수의 목적은 해당 모양의 Draw()함수를 호출하는것이다.

DrawFigure()함수는 이 과제를 완성하기 위하여SeqIterator<Shape*>형의 반복자 P를 리용한다. 목록의 각이한 요소들을 처리하는 for순환본체는

(*(*p)).Draw()

이다. (*p)는 반복자 P를 참조해제하며 Shape에 대한 지적자를 생성한다. 이 값은 그림에 있는 모양중 하나에 대한 지적자이다. (*(*p))는 지적자 (*p)를 참조해제하여 현재모양을 생성한다. 선택연산자는 Draw()성원함수가 호출될수 있도록 응용된다. Shape가 가상함수인 Draw()를 가지고 있으므로 파생클라스의 Draw()함수가 호출된다.

BuildHouse() 와 DrawFiqure() 함수는 프로그람 14-4에 있는 ApiMain()이 리용하는데

DrawHouce가 지적하는 객체를 생성하고 표시한다.

```
SeqList<Shape*>*DreamHouse = BuildHouse(Window);
DrawFiqure(DreamHouce);
```

14.9 가상적인 다중계승

virtual수식어는 또한 클라스파생과 함께 리용할수 있으며 특히 파생된 클라스가 같은 기초클라스로 계승할 때 쓸모 있다. 수식어가 적용되면 기초클라스자료성원은 복사만이 계승된다. 이 동작을 설명하기 위하여 먼저 int형자료성원 DataValue를 가진 BaseClass와 2개의 파생클라스들인 DerivedClass1와 DerivedClass2를 정의하자.

```
class BaseClass {
    protected:
        int DataValue;
};
class DerivedClass1:public BaseClass {
    // ...
};
class DerivedClass2:public BaseClass {
    // ...
};
```

MultipleClass가 DerivedClass1과 Drivedclass2에서 virtual수식어가 파생된다면 MultipleClass형의 객체는 이름이 DataValue인 2개의 서로 다른 자료성원들을 가지게 될것이다.

```
class MultipleClass1
 :public DerivedClass1, public DerivedClass2
    // ...
};
```

MultipleClass1성원함수는 두개의 자료성원들사이에 구별되는 결과연산자를 리용하여야 한다. 아래의 입력/출력조작은 지적한 DataValue객체에 대한 명백한 참조이므로 옳다.

```
// MultipleClass1 의 성원함수에서
cout << DerivedClass1::DataValue; //명백함
cin >> DerivedClass2::DataValue1; //명백함
```

다음의 대입명령문은 어느 DataValue가 참조되고 있는지 구별하는 문맥이 없으므로 MultipleClass1 성원함수에서는 틀린다.

```
//MultipleClass1 성원함수에서
DataValue = 1024;  //분명치 않음
```

클라스파생에서 virtual수식어의 효과를 보기 위하여 MultipleClass2를 BaseClass로부터 가상적으로

파생된 DerivedClass3과 DerivedClass4에서 파생한것으로 정의하시오.

```
class DerivedClass3:virtual public BaseClass {
    // ...
};
class DerivedClass4:virtual public BaseClass {
    // ...
};
class MultipleClass2 :
  :virtual public DerivedClass3,
    virtual public DerivedClass4 {
    // ...
};
```

MultipleClass2의 성원함수에서 어느 DataValue가 참조되고 있는지 구별하기 위하여 결과연산자를 리용할 필요는 없다. 즉 **virtual**수식어는 오직 하나의 자료성원 DataValue만을 가지고 있음을 확인한다. 자료성원은 직접 요소기초클라스와 하나의 결과연산자를 리용하여 참조될수 있다. 따라서 다음의 출력명령문은 같은 결과를 만든다.

```
// MultipleClass2 성원함수에서
cout << DataValue << endl;
cout << BaseClass::DataValue << endl;
cout << DerivedClass3::DataValue << endl;
cout << DerivedClass4::DataValue << endl;
```

콤퓨터의 력사

정보초고속도로

콤퓨터의 계산능력이 높아 지는것과 함께 콤퓨터망에 많은 계산기들이 망라되게 되였다. 방대한 량의 정보를 콤퓨터망을 통하여 호출할수 있다. 실례로 토마스 제퍼슨의 많은 강의는 인터네트를 통하여 할수 있게 되였다. 이 전자적인 능력은 전 세계의 연구사들이 자기 사무실을 떠나지 않고도 그 강의를 받을수 있게 한다. 이 책을 사용하는 대학의 학생들은 전자우편을 가지고 클라스에 대하여 소개한 실례나 새 소식에 접하면 Web싸이트(즉 http::/www.cs.virginia.edu/cs101)를 찾으시오. 학생들은 질문이 생기면 전화가 아니라 전자우편을 교수에게 보낼수 있으며 교수를 만나볼수도 있다. 보통 그들은 질문에 대한 대답을 받는다. 더우기 질문이 해결될 때 질문과 해답이 새소식묶음이나 Web싸이트에 적당히 부쳐 진다. 따라서 학생들은 다른 학생들이 물어 본 질문으로부터 조언을 받을수 있다. 정보에 대한 광범하면서도 값 눅은 고속접근이 사회를 어떻게 변화시키는가 하는것은 콤퓨터과학자들과 사회과학자들에게 있어서 높은 관심사의 하나이다. 우리는 확실히 흥미 있는 시대에 살고 있다.

문 제

9. 다음의 코드를 분석하시오.

```
#include <iostream>
using namespace std;
class BaseClass {
    public:
        BaseClass (int I = 3);
        virtual void Print() const;
        int GetValue() const;
    private:
        int MyValue;
};
BaseClass::Print() const {
    cout << "BaseClass Print " << GetValue() << endl;
    return ;
}
int BaseClass::GetValue() const {
    return MyValue;
}
class DerivedClass:public BaseClass {
    public:
        DerivedClass(int i =12);
        virtual void Print() const;
        int GetValue() const;
    private:
        int MyValue;
};

DerivedClass::DerivedClass(int v):MyValue(v) {
}
void DerivedClass::Print() const {
    cout << "DerivedClass Print" <<GetValue() << endl;
    return;
}
int DerivedClass::GetValue() const {
    return MyValue;
}
int main() {
```

```
    BaseClass B(1);
    DerivedClass D(10);
    BaseClass *Bptr1 = new BaseClass(3);
    BaseClass *Bptr2 = new DerivedClass(5);
    DerivedClass *Dptr1 = new DerivedClass(10);
    B.Print();
    Bptr2 -> Print();
    Bptr2 = Dptr1;
    Bptr2 -> Print();
    return 0;
}
```

이 프로그램의 출력은 무엇인가?

10. 다음의 코드를 분석하시오.

```
#include <iostream>
using namespace std;
class BaseClass {
    public:
        BaseClass(int I = 3);
        virtual void Print() const;
        int GetValue() const;
    private:
        int MyValue;
};
BaseClass::BaseClass(int v):MyValue(v) {
}
void BaseClass::Print() const{
    cout << "BaseClass Print"<<GetValue() << endl;
}
int BaseClass::GetValue() const {
    return MyValue;
}
class DerivedClass:public BaseClass {
    public:
        DerivedClass(int I = 12);
        virtual void Print() const;
        int GetValue() const;
    private:
    int MyValue;
};
```

```
DerivedClass::DerivedClass(int v):MYValue(v) {
}
void DerivedClass::Print() const {
    cout << "DerivedClass Print" << GetValue() << endl;
    return;
}
int DerivedClass::GetValue() const {
    return MyValue;
}
void f(BaseClass &B) {
    B.Print();
    return;
}
int main() {
    BaseClass B(1);
    DerivedClass D(10);
    f(B);
    f(D);
    return 0;
}
```

이 프로그램의 출력은 무엇인가?

11. 사용자가 파생클라스객체를 기초클라스객체에 할당할 때 무슨 현상이 일어나는가? 문제 9에 있는 코드의 선언에서 명령문

```
B=D;
```

은 이러한 할당의 실례이다.

12. 11번 문제의 해답을 리용하면 다음의 프로그램의 출력결과는 무엇인가?

```
#include <iostream>
using namespace std;
class BaseClass {
    public:
        BaseClass(int I = 3);
        virtual void Print() const;
        int GetValue() const;
    private:
        int MyValue;
};
BaseClass::BaseClass(int v):MyValue(v) {
}
```

```
void BaseClass::Print() const{
    cout << "BaseClass Print"<<GetValue() << endl;
}
int BaseClass::GetValue() const {
    return MyValue;
}
class DerivedClass:public BaseClass{
    public:
        DerivedClass(int I = 12);
        virtual void Print() const;
        int GetValue() const;
    private:
        int MyValue;
};
DerivedClass::DerivedClass(int v):MYValue(v) {
}
void DerivedClass::Print() const {
    cout << "DerivedClass Print" << GetValue() << endl;
    return;
}
int DerivedClass::GetValue() const {
    return MyValue;
}
int main() {
    BaseClass B(1);
    DerivedClass D(10);
    B.Print();
    D.Print();
    B = D;
    B.Print();
    return 0;
}
```

13. 기초클라스객체를 파생클라스객체로 복사하는것은 오유이다. 문제 12의 코드에 있는 선언들과 명령문

```
D=B;
```

를 리용하여 그 리유를 설명하시오.

14.10 알아 둘 점

- ✓ 다형성은 객체의 형에 따라 각이한 함수나 연산자를 호출하기 위하여 같은 대면부를 리용할수 있게 하는 언어기구이다.
- ✓ 함수다중정의에서 이름을 다시 리용하는것은 다형성의 원시적인 형식이다. 다른 방법은 함수와 클라스본보기를 리용하는것이다.
- ✓ 함수본보기는 새로운 함수를 만들기 위한 기구이다.
- ✓ 클라스본보기는 새로운 클라스를 만들기 위한 기구이다.
- ✓ 본보기파라메터에는 형과 값이 있다.
- ✓ 함수본보기정의에서 형본보기파라메터들은 발생한 함수의 되돌림형과 파라메터형을 규정하는데 리용된다.
- ✓ 모든 본보기파라메터는 함수대면부에서 리용되여야 한다. 본보기파라메터들은 또한 함수본체에서 리용될수 있다.
- ✓ Ctr는 부분탐색과 순차배렬과 같은 계산과제의 본보기를 가진 표준본보기서고들을 제공한다.
- ✓ 클라스본보기에서 모든 본보기파라메터들은 클라스대면부정의에서 리용되여야 한다.
- ✓ 클라스본보기는 용기클라스 ADT를 작성하는데서 매우 쓸모 있다. 용기 ADT는 객체목록을 나타낸다.
- ✓ 용기 ADT를 분류하는 기본방법은 목록에 있는 요소들에 대하여 임의접근을 제공하는가 아니면 순차접근을 제공하는가에 관계된다.
- ✓ 클라스본보기를 리용하여 배렬과 같이 목록을 나타내는 용기클라스를 작성할수 있다. 그런 용기클라스가 표준배렬보다 유리한것은 용기클라스에서 배렬이 제한되지 않기때문이다(실례로 용기클라스는 함수의 되돌이형이 될수도 있고 값파라메터일수도 있다).
- ✓ 련결목록은 값들의 동적목록을 실현하는 한가지 방법이다. 련결목록은 두개의 구성요소를 가지는 객체들의 묶음을 리용하여 집합을 나타낸다. 한개 요소는 나타내고 있는 목록값을 그대로 유지하며 다른 요소는 목록에 있는 다른 객체에 대한 하나이상의 지적자이다. 보통 존재하는 지적자는 다음객체를 지시하는 지적자이다. 2중련결목록에서 다른 지적자는 선행자객체를 지시하는 지적자이다.
- ✓ 클라스 friend는 함수, 연산자, 다른 클라스일수도 있다. 클라스 friend는 그 클라스의 자료성원들을 모두 호출할수 있다. 이런 호출은 정보은페원리에 맞지 않는다.
- ✓ 표준본보기서고는 목록을 나타내는 일반용기클라스들의 집합을 정의한다. 목록의 요소들이 어떻게 호출될수 있는가에 따라 표현방법이 다르다. 여러가지 용기클라스들은 요소에 대한 임의접근과 순차접근 그리고 혼합방식을 정의한다.
- ✓ 용기클라스는 그와 련관된 반복자클라스를 가진다. 반복자클라스는 목록의 각이한 요소들을 반복 호출하는 기능을 제공한다.
- ✓ 변환연산자들은 재정의될수 있다. 다중정의할 표준변환연산자는 bool이다. 규정된 자료형을 가진 객체에 대하여 변환이 적용될때 변환은 그 객체가 요구하는 값을 가지는가를 의미하는 값을 만든다. 실례로 SeqIterator반복자클라스에서 bool연산자는 반복자 객체가 현재목록안에 있는 유효한 요소를 나타냈는가를 지적하는데서 재정의된다.
- ✓ C++에서 다형성을 실현하는 기본방법은 가상함수를 리용하는것이다.
- ✓ 가상함수는 성원함수이여야 한다.

✓ 파생클라스의 함수가 기초클라스의 가상함수와 이름, 형이 같으면 파생클라스의 성원함수는 기초클라스함수를 재정의한다.
✓ 가상함수를 리용하면 실지 호출될 함수의 결정이 실행시까지 지연된다. 지적자나 참조객체의 호출되는 객체형에 따라 함수가 결정된다.
✓ 지적자배렬을 선언하여 여러가지 목록을 나타내는데 리용할수 있다. 즉 개별적인 요소들은 기초클라스로부터 서로 다른 파생클라스를 지적할수 있다.
✓ 가상함수가 여러가지 목록에 있는 요소들중 하나를 참조해제하여 호출될 때 동작은 그 지적자가 가리키는 객체형에 따라 다르다. 새롭게 파생된 형이 정의되고 목록에 추가되여도 정확히 작업을 계속할것이다.
✓ 기초클라스의 해체자는 보통 가상함수이다. 왜냐하면 문맥파괴에 관계없이 적당한 해체자가 호출되여야 하기때문이다.
✓ 순수가상함수는 null주소가 할당된 가상함수이다.
✓ 순수가상함수에는 실현부가 없다.
✓ 순수가상함수를 가진 클라스로부터 객체를 구축할수 없다. 리유는 구축이 완성될수 없기때문이다.
✓ 순수가상함수를 가진 클라스는 가상기초클라스이다.
✓ 가상기초클라스는 자기의 파생클라스의 일반적대면부를 정의하는데 리용된다. 실례로 Shape조작의 일반동작이 도형을 그리는것이며 가상성원함수 Draw()가 Shape의 공통대면부의 한 부분이여야 한다. 파생된 Shape만이 묘사될수 있는 충분한 특성을 가지므로 Draw()성원은 순수가상함수여야 한다.
✓ **virtual**수식어는 클라스파생과 함께 리용될수 있다.
✓ **virtual**수식어는 파생클라스가 같은 기초클라스로부터 여러번 계승될 때 효과적이다. 수식어는 그 기초클라스로부터 매 자료성원을 한번만 복사하여야 한다. 수식어가 없으면 표준규칙이 적용되는데 이것은 매개 계승된 클라스가 자료성원의 복사를 개별적으로 지원한다는것을 의미한다.

련습문제

14.1 본보기가 객체지향문법에서 가상함수만큼 중요하게 고려되지 않는 리유를 설명해 보시오.
14.2 본보기가 순수한 다형성이라기보다 함수재정의에서 이름의 재리용이라고 고찰되는 원인을 설명하시오.
14.3 어느 가상함수가 호출되는가가 실행시에 결정되는 리유는 무엇인가?
14.4 배렬에 대한 어떤 특정한 값이 있는가를 결정하는 검색함수의 본보기를 작성하시오. 본보기파라메터의 형인 본보기용기클라스 Array를 리용하여 본보기함수가 동작할수 있는가 ? 그 리유는 무엇인가?
14.5 QuickSort()의 본보기를 작성하시오.
14.6 SeqList<T>객체로 표현되는 목록에서 최대, 최소값을 찾는 Min(), Max()의 본보기함수를 설계하시오.
14.7 다음의 정의에서 오유를 찾아 내고 수정하시오.

```
template<type s, type T>
int S f(s  A[n]) {
    S x;
    cin >> x;
```

```
    A[x] = x;
}
```

14.8 주어 진 수가 SeqList<T>객체가 표시한 목록에 있는가를 결정하는 본보기함수 Search()를 작성하시오. 주어 진 값이 목록에 있으면 그 값을 가진 요소와 련관된 SeqIterator()반복자를 되돌려야 한다. 없으면 《보호원소》와 련관된 SeqIterator<T>반복자를 되돌려야 한다.

14.9 SeqList<T>객체로 표현된 목록을 순차배렬하는 본보기함수 Sort()를 작성하시오.

14.10 **friend**란 무엇인가? 그의 우점은 무엇인가?

14.11 Rational클라스를 그 자료성원들의 형이 옹근수값을 표시하는 본보기로 되도록 수정하시오. 여러 가지 표준자료형들을 리용하여 시험해 보시오.

14.12 본보기클라스 Bunch<T, n>을 작성하시오.

14.13 2개의 파라메터 n과 val을 가진 Array<T>성원함수 resize()를 작성하시오. 파라메터 n은 목록의 크기를 나타내는 옹근수값이고 기정값으로는 0이다. Val의 파라메터는 목록에 추가되는 객체값을 나타내는 상수 T객체에 대한 참조이다. 그의 가상값은 T의 객체값과 같다. 낡은 목록에 있는 요소는 그대로 남아 있어야 한다. 이 조작의 실행시간은 무엇인가?

14.14 Vector push_Back()함수를 모의하는 Array<T>성원함수 push_back()를 작성하시오. 이 조작의 실행시간은 무엇인가?

14.15 Vector clear()함수를 모의한 Array<T>성원함수 clear()함수를 작성하시오. 이 조작의 실행시간은 무엇인가?

14.16 vector insert()함수를 모의한 Array<T>성원함수 insert()를 작성하시오. 이 조작의 실행시간은 무엇인가?

14.17 vector insert()성원함수를 모의한 Array<T>성원함수 eraser()를 작성하시오. 이 조작의 실행시간은 무엇인가?

14.18 의뢰자가 요구하는 간격을 규정할수 있도록 Array<T> ADT를 다시 작성하시오. 실례로 Array<**int** >Month(1, 12, 0);은 12개 요소를 가진 배렬이다. 첫 요소의 첨수번호는 1, 마지막 요소는 12이다. 모든 요소들은 다 0으로 초기화된다.

14.19 Array<T>와 SeqList<T>를 적당한 계승을 위한 기초클라스로 만드는데서 필요한것은 무엇인가?

14.20 SeqList<T>객체에 대하여 복사구축자나 성원값주기조작이 규정되지 않는다. 가상적으로 어떤 형의 조작이 실행되는가? 왜 그런가? 명백하게 2개의 성원조작의 복사를 만드시오.

14.21 연산수 i가 목록에서의 번호인 성원첨자연산자 []를 포함하여 SeqList<T> ADT를 수정하시오. 연산자는 i번째 요소에 대한 참조를 돌려 준다. Array<T>와 SeqList<T>의 []에 대하여 연산자의 상대적인 효과성을 비교해 보시오.

14.22 아래의 코드의 결과는 무엇인가? 왜 그런가 ?

```
SeqList<int> P;
P.push_back(1);
P.push_back(2);
P.push_back(3);
SeqIterator<int> a = P.begin();
SeqIterator<int> b = P.begin();
++a;
++a;
*a = *b;
```

```
P.display(cout);
```

14.23 SeqIterator<T>의 반복자는 왜 SeqList<T>클라스부분이 아니라 개별적인 클라스에서 작성하는가?

14.24 SeqList<T> 반복자ADT를 다시 설계하여 성원함수 insert_after()가 정의되도록 하시오. 함수는 2개의 파라메터 즉 반복자 p와 값 val을 가진다. 값 val의 복사는 반복자 p가 지적하는 요소다음에 추가된다.

14.25 반복자클라스 SeqIterator<T>를 리용하도록 SeqList<T>성원함수 Display()를 작성하시오.

14.26 반복자클라스 SeqIterator<T>를 리용하도록 SeqList<T>성원함수 clear를 다시 작성하시오.

14.27 SeqList<T>객체용추가연산자를 설계, 작성하시오. 연산결과는 오른쪽연산수 SeqList<T>에 표시된 요소의 복사를 SeqList객체의 목록의 마지막에 추가하는것이다.

14.28 SeqList<T>객체의 삭제연산자를 설계, 작성하시오. 연산자는 오른쪽연산수 SeqList<T>에 의하여 모든 요소들을 목록에서 삭제해야 한다.

14.29 SeqList<T>성원함수 Display()를 리용하여 SeqList<T>객체용출력연산자를 다중정의하시오.

14.30 련결목록에 요소가 없으면 참을 되돌리는 SeqList<T>객체의 Bool형 Empty()성원함수를 설계, 작성하시오.

14.31 자료성원 ListLength가 없이 SeqList<T>를 다시 작성하시오. 기능이 없어 지면 안된다. 어느 성원이 실행시간에 영향을 주며 어떻게 하는가?

14.32 SeqIterator<T>용출력연산자를 재정의하는데서 반복자가 실지요소와 련결되였는지 확인할 필요가 왜 없는가?

14.33 SeqIterator<T>의 앞붙이연산자 --를 작성하시오.

14.34 SeqIterator<T>의 뒤붙이연산자 --를 작성하시오.

14.35 SeqIterator<T>의 **bool**형성원함수 Isfront()를 반복자가 목록의 첫 요소와 련관되면 **true**를 되돌리도록 작성하시오.

14.36 SeqIterator<T>의 성원함수 isback()를 반복자가 목록의 마지막요소와 련관되면 참을 돌려 주도록 작성하시오.

14.37 SeqIterator<T>의 클라스를 가상구축자가 2개의 선택적파라메터인 L과 Pos를 가지도록 다시 작성하시오. 파라메터 L은 반복자와 련관된 목록의 지적자이다. L의 기정값은 null주소이다. 파라메터 pos는 IterStatus형인데 이것의 정의는 다음과 같다.

```
enum IterStatus
{frontposition, backposition, sentinelposition};
```

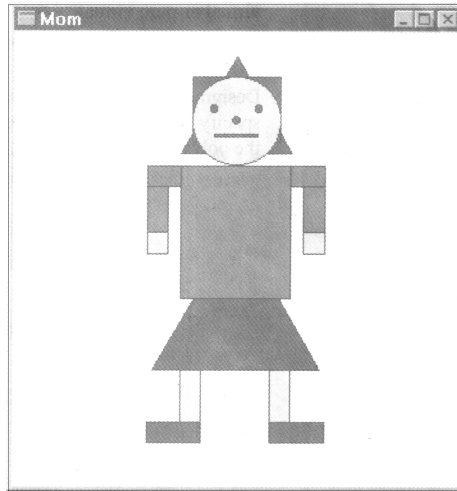
pos의 기정값은 sentinelposition이다. pos의 값이 frontposition이면 반복자가 목록의 첫 요소와 련관된다. pos의 값이 backposition이면 반복자는 목록의 마지막요소와 련관된다. sentinelposition이면 반복자는 목록의 보호원소와 련관된다. 이 경우에 지정된 요소가 있는지 검사하여야 한다.이런 구축자를 리용할수 있도록 SeqList<T> 코드를 수정하시오. 왜 이런 노력을 하여야 하는가?

14.38 상수 SeqList<T>객체가 호출될수 있도록 SeqIterator<T>클라스를 설계, 작성하시오. 가능한껏 SeqIterator<T>클라스의 특성을 다 반영하시오. 또한 실행은 SeqIterator<T>객체로부터**const** SeqIterator<T>객체를 구축하는 구축자를 포함하여야 한다. 보충적인 구축자가 **const** SeqIterator <T>객체로부터 SeqIterator<T>객체를 구축하는 seqIterator<T>클라스에 추가되여야 하는가? 왜 그런가?

14.39 우의 문제에 있는 ConstSeqIterator<T>클라스를 리용하여 SeqList<T>::begin()와

SeqList<T>::end()를 const성원함수로 작성하시오.

14.40 기초용기클라스본보기 Bucket를 작성하시오. Array<T>와 SeqList<T>클라스는 bucket<T>로부터 파생되여야 한다. SeqIterator<T>클라스는 Array<T>와 SeqList<T>와 함께 동작하도록 다시 설계하고 이름을 다시 달아 주어야 한다.

14.41 가상기초클라스 Shape와 사람을 그리는 파생클라스를 리용하는 BuildPerson()함수를 BuildHouse()와 비슷하게 작성하시오. 그림은 아래에 주었다.

14.42 왜 즉시 되돌이하는 가상함수가 아니라 순수가상함수를 리용하는가를 생각해 보시오. 파생클라스가 정의를 재정의하지 않는다면 어떤 현상이 나타나는가?

14.43 추상기초클라스 Shape에서 가상함수로서 실현할수 있는 동작을 고르시오. 왜 그런가?

a) 회전
b) 재배치
c) 확대 및 축소
d) 이동
e) 채우기문양설정

14.44 우의 목록을 가지고 가상기초클라스 Shape에서 순수가상함수로서 실현할수 있는 동작을 고르시오. 왜 그런가?

14.45 13장에서 론의한 필기도구계층구조를 작성하기 위한 추상기초클라스를 설계하시오. 여러가지 형식의 기구를 작성하기 위한 파생클라스를 작성하시오. 여러 성원함수들과 련관된 기구속성을 지정하도록 삽입명령문을 사용하시오.

14.46 운수기재의 계층구조를 설계하시오. 기초클라스로부터 연료 즉 석탄, 태양열, 화학, 증기, 핵 등에 따라 또한 상업, 개인, 농업, 정치, 등 사용목적에 따라, 운수기재가 바퀴형식인가, 아닌가에 따라 운수기재를 계층화하시오. 그리고 썰매차, 통학뻐스, 짐차 등을 위한 클라스계층구조를 구성하시오. 어느 함수가 가상함수이며 어느 함수가 순수가상함수인가?

14.47 나무를 표시하는 계층구조를 설계하시오. 어느 함수가 가상함수이며 어느 함수가 순수가상함수인가?

14.48 새를 표시하는 계층구조를 설계하시오. 어느 함수가 가상함수이며 어느 함수가 순수가상함수인가?

14.49 EzWindows클라스 Position을 리용하여 행의 끝점을 지정하도록 Segment클라스를 설계하시오. Segment객체목록을 리용하여 다각형을 그리는 모형계층구조를 다시 정의하시오.

제 15 장. 쏘프트웨어대상과제-벌레잡이

소개

이제는 지금까지 배워 온 객체지향적인 경험을 발휘할 때가 되였다. 이 장에서는 《벌레잡이》유희를 설계하고 실현한다. 유희의 목적은 창문주위를 돌아 가는 벌레들을 없애는것이다. 벌레는 마우스로 찰칵하여 없앤다. 유희는 계층, 가상함수 그리고 다형성을 비롯한 C++의 객체지향적인 특징들을 사용하여 실현한다. 유희에서는 EzWindows API를 많이 사용한다.

기본개념

- 교갑화
- 계승
- 파생클라스
- 가상함수
- 다형성
- 객체지향설계

15.1 벌레잡이

벌레잡이(Bug Hunt)는 간단한 놀이이다. 유희의 목적은 창문에서 돌아 다니는 벌레를 없애는것이다. 벌레는 마우스로 찰칵하여 없앤다. 프로그람에서의 오유와 같이 Bug Hunt의 벌레는 없애기가 힘들다. 벌레를 없애려면 벌레를 여러번 눌러야 한다. 벌레를 없애면 즉시 다른 벌레들이 나타난다. 유희는 없애버릴 벌레가 더이상 없을 때 끝나게 된다. 만일 사용자가 벌레를 놓치면 유희는 다시 시작된다.

간단히 유희의 고수준설계를 시작하자. 유희는 여러 객체들로 구성된다. 기본객체가 벌레라는것은 앞에서 명백히 하였다. 사실 유희는 두가지 형식(벌레잡이가 쉬운 낮은 급과 벌레잡이가 힘든 높은 급)을 가진다. 계승을 리용하여 벌레들을 간단히 실현할수 있다. 다른 객체는 벌레들이 있는 창문이다. 창문은 EzWindow클라스 SimpleWindow로 실현된다. 또한 유희를 조종하기 위한 객체인 유희조종자가 있어야 한다. 유희조종자는 유희를 설정하고 유희의 상태를 유지하며 유희과정을 조종한다. 그림 15-1에 프로그람의 고수준설계를 보여 주었다. EzWindows를 사용하여 유희조종자들은 창문으로부터 마우스찰칵사건과 박자발생사건을 받는다. 마우스를 누를 때 유희조종자는 지정된 벌레에게 통보를 보낸다. 이때 벌레는 맞은것으로 된다. 일정한 정도로 맞으면 벌레는 죽는다. 벌레가 죽으면 유희조종자는 벌레를 제거하고 다음단계로 넘긴다. 마우스지시기가 벌레를 가리키지 않은 상태에서 마우스가 눌리웠다면 유희조종자는 유희가 계속되도록 한다.

박자발생사건때 유희조종자는 벌레가 움직일것을 지시한다. 박자발생간격이 일정한 정도로 좁다면 벌레는 창문안의 주위를 도는것처럼 보일것이다. 물론 벌레가 어떻게 움직이는가 하는것은 벌레에 대한 문제이며 유희조종자는 그것을 알 필요도 리해할 필요도 없다. 이 프로그람은 3개 모듈로 구성된다.

bug.cpp모듈은 벌레의 실행과 관계되는 코드부분이다. control.cpp는 유희조종자를 실현한 모듈이며 bughunt.cpp에는 프로그람의 시작과 완료에 관한 코드가 포함한다(즉 ApiMain()와 ApiEnd()). 더우기 매 모듈에는 대면부를 정의하는데 대응되는 .h파일이 있다. 물론 이 프로그람은 EzWindows API서고코드와 관련되게 된다.

유희에서 벌레의 행동이 기본이므로 그것부터 시작하자 .

15.2 기초클라스 BUG

유희는 여러가지 형식의 벌레를 요구하므로 계층을 리용하여야 한다. 기본의도는 기초클라스에 벌레의 일반적움직임을 모두 포함시키고 계승을 통하여 개별적인 벌레(즉 각이한 동작을 한다.)를 만든다는것이다.

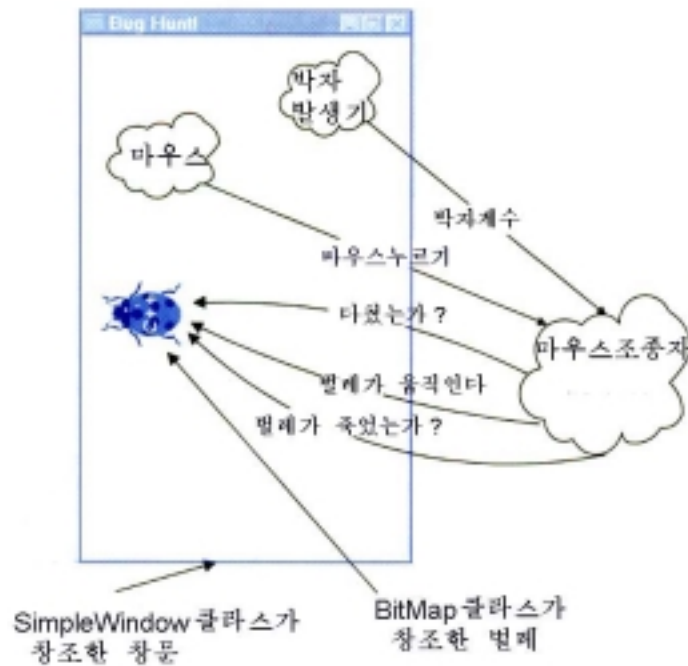



그림 15-1. Bug Hunt의 고수준설계

그림 15-2에서 보는것처럼 기초클라스 Bug에서 SlowBug와 FastBug가 파생된다. SlowBug와 FastBug의 차이는 그것들이 어떻게 움직이는가 하는것이다.

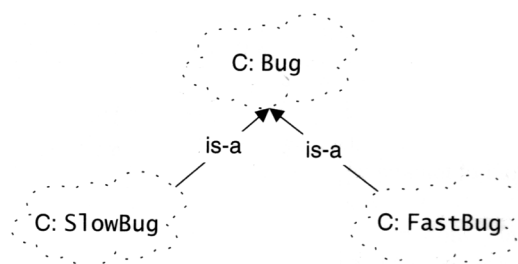



그림 15-2. Bug로부터 파생된 SlowBug와 FastBug

유희조종자의 견지로부터 두가지 형태의 벌레들의 대면부는 같아야 한다. 따라서 유희조종자의 견지로부터 벌레에게 필요한 동작들을 고려하여 설계를 시작하자! 그림 15-1에서 보는것처럼 유희조종자는 벌레를 맞혔는가, 벌레가 이동했는가, 벌레가 죽었는가를 아는것이 필요하다. 또한 유희조종자는 유희가 동작할 때 벌레들이 죽었는가를 알아야 한다.

간단히 말하면 벌레의 공개대면부가 포함되여야 한다.

- Move - 벌레를 창문에서 다음 위치로 이동시킨다.
- IsHit - 제공된 창문자리표에 벌레가 있는가를 결정한다.

조건이 맞으면 맞힌 수를 결정하고 참을 돌려 주며 그렇지 않으면 거짓을 돌려 준다.

- IsDying - 벌레를 맞힌 수가 벌레가 죽는데 필요한 회수와 같거나 크면 참을 되돌린다.
- Create - 벌레를 창조하고 창문에 그린다.
- Kill - 벌레가 창문에서 제거된다.

Bug에는 어떤 속성이 있어야 하는가? 벌레가 EzWindows객체인 SimpleWindow에 그려 지므로 그 창문에 대한 참조를 가지고 있어야 한다. 또한 창문안에서의 위치가 있어야 한다. Bug는 벌레의 화상을 표시하기 위해 EzWindows BitMap도 포함한다. 창문에서 벌레의 움직임은 4개의 비트화상을 리용하여 보게 된다. 매 화상은(그림 15-3) 이동한 벌레의 방향에 해당된다. Bug는 벌레의 방향을 변화시킬수 있는 속성을 가지고 있다. 그것은 벌레의 변화가 운동처럼 보이기때문이다.

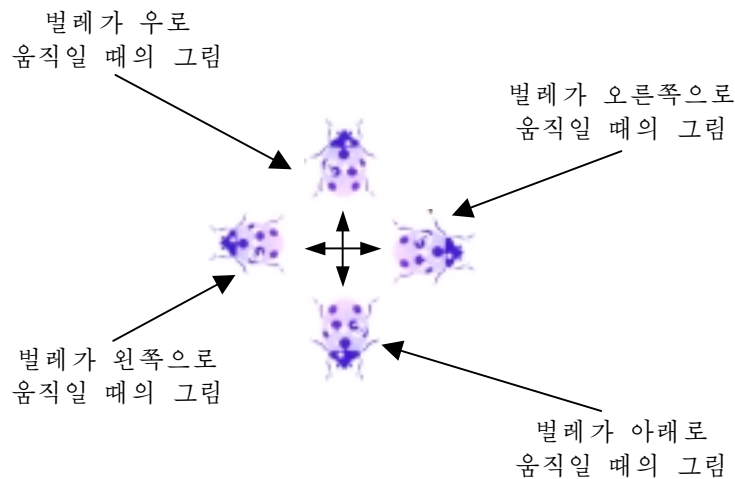



그림 15-3. 움직이는 벌레를 그릴 때 리용하는 비트매프

다른 속성에는 현재 움직인 벌레의 방향과 지금까지 마우스로 벌레를 때린 수, 벌레를 죽이기 위해 마우스로 때린 수가 있다. 간단히 말하면 벌레의 속성에는 다음과 같은것이 있어야 한다.

- Window: 벌레가 있는 창문
- Bmp: 벌레의 방향에 대응하는 4개의 비트매프배렬
- HitsTaken: 벌레를 찰칵한 회수
- HitsRequired: 벌레를 죽이는데 필요한 찰칵회수
- DirectionChangeProbability: 벌레가 방향을 바꿀수 있는 가능성
- CurrentDirection: 벌레가 움직이는 방향
- CurrentPosition: 벌레의 현재의 위치

중요한 문제는 움직이는 벌레를 창문에 어떻게 나타내겠는가 하는것이다. 어떤 객체가 움직이듯이 보이게 하는 간단한 방법은 그 객체의 비트매프를 지우고 새 위치에 다시 그리는것이다. 그림 15-4에 이 방법을 주었다. 지우기/다시 그리기조작을 계속하며 새 위치가 초기위치로부터 너무 멀지 않다면 화상은

움직이는것처럼 나타날것이다. 객체들이 많지 않고 그것들이 그렇게 빨리 움직이지 않는다면 이 방법이 적합하다. 더 원활하게, 더 빨리 움직일것을 요구한다면 다른 방법들을 사용할수 있다. 그러나 이 유희에서는 간단한 방법을 적용할수 있다.

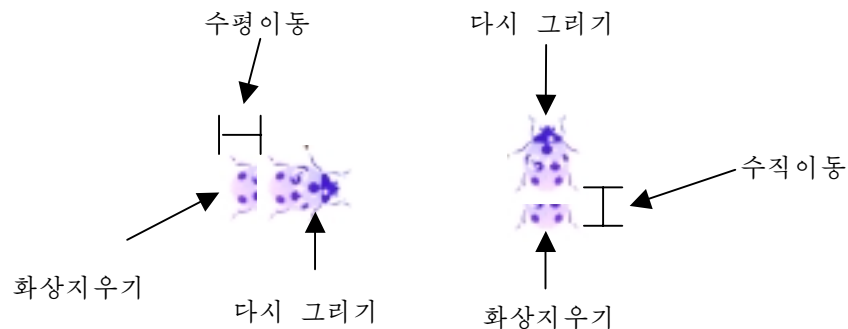



그림 15-4. 지우기와 다시 그리기에 의한 이동

Bug클라스에는 또한 2개의 자료성원인 HorizMovement와 VertMovement가 있다. 이것들은 각각 비트매프의 수평 및 수직이동량을 나타낸다.

물론 공개성원함수들외에 Bug클라스에는 검토자와 변이자가 필요하다. 이것들은 Bug클라스와 그 파생클라스에 의해서만 리용되므로 공개대면부로 선언되지 않는다. Bug의 사용자에게 접근할수 없는 이러한 성원함수를 만들며 Bug로부터 파생된 클라스들게 접근하도록 하자면 보호부분으로 선언되여야 한다. 따라서 Bug클라스는 공개대면부와 보호대면부를 가진다. Bug클라스의 자료성원들은 비공개부분이며 파생클라스는 그것들에 접근하기 위하여 보호부분에서 제공되는 검토자와 변이자를 리용하여야 한다. 목록 15-1에 Bug클라스를 주었다.

목록 15-1. bug.h 에 있는 Bug클라스의 선언

```
#ifndef BUG_H
#define BUG_H
#include "randint.h"

// 벌레 한마리에 대한 클라스
enum Direction { Up, Down, Left, Right };
const int BugBitMaps = 4;    //매 방향에 대하여 2진도형이 하나씩 대응된다
class Bug {
    public:
        Bug(SimpleWindow &w, int HitsNeeded = 3,
        int DirectionChangeProbability = 50);
        bool IsHit(const Position &MousePosition);
        bool IsDying();
        void Create();
        void Kill();
        virtual void Move() = 0;
```

```
    protected:    // 검토자
        SimpleWindow& GetWindow() const;
        Position GetPosition() const;
        Direction GetDirection() const;
        float GetHorizMovement() const;
        float GetVertMovement() const;
        int GetDirectionChangeProbability() const;
        BitMap &GetBmp(const Direction &d);
        const BitMap &GetBmp(const Direction &d) const;
        // 변이자
        void SetWindow(SimpleWindow &w);
        void SetDirection(const Direction &d);
        void SetHorizMovement(float h);
        void SetVertMovement(float v);
        void Draw();
        void Erase();
        void SetPosition(const Position &p);
        void ChangeDirection();
        Position NewPosition() const;
        RandomInt GeneratePercentage;
    private:    //성원자료
        SimpleWindow &Window;
        vector<BitMap> Bmp;
        float HorizMovement;
        float VertMovement;
        int HitsRequired;
        int HitsTaken;
        int DirectionChangeProbability;
        Direction CurrentDirection;
        Position CurrentPosition;
};
#endif
```

목록 15-1에서 Bug의 선언에 앞서 4개 방향에 대한 기호이름들을 주는 enum형객체를 선언하였다. 벌레의 방향코드안의 번호대신 렬거형을 사용하면 프로그람을 더 잘 리해할수 있다. 공개성원함수 Move()가 순수가상함수이기때문에 Bug는 추상기초클라스이다. 이 벌레들은 움직이는 방법이 다르다. Bug에 순수가상함수를 선언하면 Bug형객체를 리용할수 없다.

Bug의 부분성원함수들은 짧고 간단하다. 목록 15-2에 Bug클라스에 대한 성원함수의 실현부를 주었다.

목록 15-2.　　　bug.cpp에 있는 Bug클라스의 구축자와 성원함수들

```
//Bug():구축자
Bug::Bug(SimpleWindow &w,int h, int p) : Window(w), HitsRequired(h),
  HitsTaken(0), GeneratePercentage(1, 100), DirectionChangeProbability(p) {
    Bmp.reserve(BugBitMaps);
    GeneratePercentage.Randomize();
    return;
}
void Bug::Create() {
    HitsTaken = 0;
    Draw();
    return;
}
void Bug::Kill() {
    Erase();
    return;
}
// Hit():마우스로 벌레를 누르면 "참"을 되돌리고 찰칵회수를 변경시킨다
bool Bug::IsHit(const Position &MousePosn) {
    if(GetBmp(GetDirection()).IsInside(MousePosn)) {
        ++HitsTaken;
        return true;
    }
    else
        return false;
}
//벌레가 죽었는가
bool Bug::IsDying() {
    return HitsTaken >= HitsRequired;
}
//검토자
SimpleWindow& Bug::GetWindow() const {
    return Window;
}
Position Bug::GetPosition() const {
    return CurrentPosition;
}
Direction Bug::GetDirection() const {
    return CurrentDirection;
```

```
}
float Bug::GetHorizMovement() const {
    return HorizMovement;
}
float Bug::GetVertMovement() const {
    return VertMovement;
}
int Bug::GetDirectionChangeProbability() const {
    return DirectionChangeProbability;
}
BitMap &Bug::GetBmp(const Direction &d) {
    return Bmp[d];
}
const BitMap &Bug::GetBmp(const Direction &d) const {
    return Bmp[d];
}
// 변이자
void Bug::SetWindow(SimpleWindow &w) {
    Window = w;
    return;
}
void Bug::SetDirection(const Direction &d) {
    CurrentDirection = d;
}
void Bug::SetHorizMovement(float h) {
    VertMovement = h;
    return;
}
void Bug::SetVertMovement(float v) {
    VertMovement = v;
    return;
}
// 촉진자
void Bug::Draw() {
    GetBmp(GetDirection()).Draw();
    return;
}
void Bug::Erase() {
  GetBmp(GetDirection(,)Erase();return;
```

```
    void Nug::ChangeDirection(){
        RandomInt R(Up, Right);
        SetDirection((Direction) R.Draw());
        return;
    }
    // 위치설정
    void Bug::SetPosition(const Position &p) {
        for (Direction d = Up; d <= Right; d = (Direction)(d + 1))
            Bmp[d].SetPosition(p);
        CurrentPosition = p;
        return;
    }
    // 새 위치계산
    Position Bug::NewPosition() const {
        const Position OldPosition = GetPosition();
        if(GetDirection() == Left)
            return OldPosition + Position(-GetHorizMovement(),0);
        else if (GetDirection() == Right)
            return OldPosition + Position(GetHorizMovement() ,0);
        else if (GetDirection() == Up)
            return OldPosition + Position(0, -GetVertMovement());
        else
            return OldPosition + Position(0, GetVertMovement());
}
```

Bug의 구축자실현부는

```
Bug::Bug(Simple Window &w,int h, int p):
  Window(w),Hits Required(h),Hits Taken (0),
  Generate PerCentage(1,100),Direction change Probability(p) {
    Bmp.reserve(Bug BitMaps),
    return;
}
```

이다. 구축자는 Bug의 자료성원들을 초기화한다. Window자료성원들은 SimpleWindow의 참조로 초기화된다. HitsRequired는 벌레를 죽이는데 필요한 찰칵회수로 초기화된다. 그리고 HitsTaken을 0으로 초기화한다. 또한 GeneratePercentage객체는 1부터 100까지의 임의의 옹근수를 설정한다. 이 모조란수렬은 벌레가 움직이는 방향을 임의로 바꾸는데 리용된다. 마지막으로 DirectionChangeProbability자료성원을 초기화한다. 이 속성은 벌레의 방향이 어느 정도 자주 바뀌는가를 조종한다.

자료성원초기화목록에서는 구추축자에 의하여 대부분 작업이 진행된다. 함수본체에 남아 있는 동작

은 Bmp벡토르에서 BugBitmaps요소를 보관하는것이다. 이 벡토르는 벌레의 비트매프화상을 표현한다. 공개성원함수 Create()와 Kill()은 보호성원함수 Draw()와 Erase()를 호출한다. 그밖에 HitsTaken자료성원을 0으로 설정한다. 이 공개성원함수의 실현부는 다음과 같다.

```
void Bug::Create() {
    Hits Taken=0;
    Draw();
    return;
}
void Bug::Kill() {
    Erase();
};
```

Bug의 보호성원함수 Draw()에서는 벌레가 움직이는 방향을 얻고 Bitmap::Draw()를 호출하여 해당한 비트매프를 그린다. 그 실현은 다음과 같다.

```
void Bug::Draw() {
    Get Bmp(GetDirection()).Draw();
    return;
};
```

Bug::Erase()의 코드는 BitMap::Erase()함수를 호출한다는 점을 내놓고는 같다.

```
void Bug::Erase() {
    Get Bmp(GetDirection()).Erase();
    return;
}
```

SetPosition()성원함수는 벌레의 위치를 설정한다. SetPosition()성원함수는 또한 4개의 방향으로 움직이는 벌레의 화상들을 포함한 4개의 비트매프위치도 설정한다.

```
void Bug::SetPosition(const Position &p) {
    for (Direction d=UP; d<=Right; d=(Direction) (d+1))
        Bmp[d].SetPosition(p);
    CurrentPosition = p
    return;
}
```

코드에서는 4개의 비트매프들을 위한 BitMap::SetPosition()을 호출한다. 그리고 Bug::CurrentPosition을 갱신한다.

NewPosition()성원함수는 벌레의 다음위치를 계산하고 돌려 준다. 함수는 벌레가 이동하는 방향을 보고 벌레의 방향에 따르는 벌레의 x자리표, y자리표의 적당한 운동거리를 더하여 현재위치를 계산한다. 그의 실현부는

```
Position Bug ::NewPosition() const {
    const Position OldPosition=GetPosition();
```

```
        if (GetDirection()==Left)
            return OldPosition+Position(-GetHorizMovement(),0);
        else if (Get Direction()==Right)
            return OldPosition+Position(GetHorizMovement(),0);
        else
            return OldPosition+Position(0, GetVertMovement()); (GetDirection()==UP));
    }
```

이다. 함수는 보호검토자 GetHorizMovement()와 GetVertMovement()를 리용하여 각각 x, y방향에서 벌레의 이동한 거리를 얻는다.

15.2.1 파생클라스 SlowBug

Bug로부터 벌레의 다른 형들, 즉 다르게 움직이는 벌레들을 만들수 있다. 그것들은 다르게 보아야하지만 그것은 문제가 아니다. 벌레잡이유희는 벌레가 밝은 청색을 가진 방향이 크게 변하지 않는 느린 벌레로부터 시작한다. SlowBug의 선언클라스는 다음과 같다.

```
class SlowBug:public Bug {
    public:
        SlowBug(SimpleWindow &w, int HitsNeeded=4, int Directionchange=10);
        void Move()
};
```

즉 SlowBug는 Bug의 한 종류이며 Bug의 모든 속성들과 동작을 계승한다. SlowBug의 구축자는 벌레를 죽이는데 필요한 찰각회수를 4로 설정한다. SlowBug는 자기 자체의 이동함수를 가진다. 이 함수는 느린 벌레의 동작을 정의한다. SlowBug클라스는 bug.h에서 Bug클라스다음에 선언하였다.

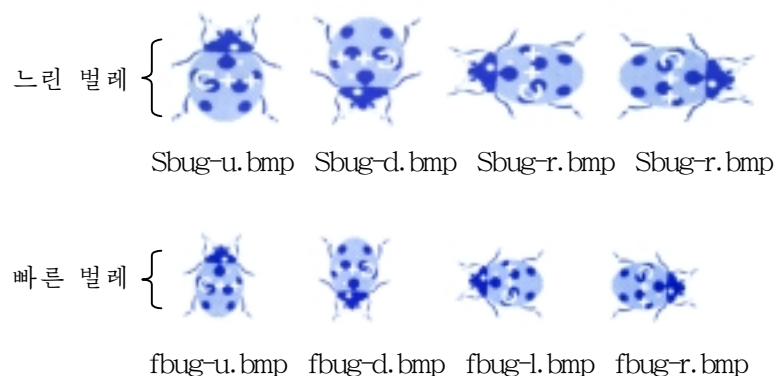



그림 15-5. 느린 벌레와 빠른 벌레 그리고 대응한 파일이름

SlowBug구축자는 Bug와 완전히 다른 동작을 한다. 특히 유희창문에서 벌레를 표시하는 비트매프들을 적재한다. 목록 15-3에 SlowBug구조체를 주었다.

목록 15-3. bug.cpp에 있는 SlowBug 구축자

```
SlowBug::SlowBug(SimpleWindow &w, int h, int p):Bug(w, h, p) {
    // 네 방향에 해당한 벌레비트도형들을 적재한다.
```

```
    vector<string> BitMapFiles(BugBitMaps);
    BitMapFiles[0] = "sbug-u.bmp";
    BitMapFiles[1] = "sbug-d.bmp";
    BitMapFiles[2] = "sbug-l.bmp";
    BitMapFiles[3] = "sbug-r.bmp";
    for (Direction d = Up; d <= right; d = (Direction)(d + 1)) {
        GetBmp(d).SetWindow(GetWindow());
        GetBmp(d).Load(BitMapFiles[d]);
        assert(GetBmp(d).GetStatus() == BitMapOkay);
    }
    // 수직, 수평방향으로 벌레가 움직일 거리를 설정한다
    // 거리는 비트매프의 크기에 의존한다
    SetHorizMovement(GetBmp(Right).GetWidth() / 10.0);
    SetVertMovement(GetBmp(Up).GetHeight() / 10.0);
    // 초기에 벌레를 오른쪽으로 가게 한다
    SetDirection(Right);
    SetPosition(Position(3.0, 3.0));
    return;
}
```

코드의 첫 부분에서는 벌레가 움직이는 4개의 방향에 해당한 비트매프들을 적재한다(그림 15-5를 보시오). 코드의 두번째 부분은 시계의 매 박자마다 벌레가 얼마만큼 이동하여야 하는가를 계산한다. 수평, 수직으로 이동한 거리를 계산하였다. 이동한 거리는 비트매프의 크기에 의존한다. 여기서 SlowBug는 매번 비트매프의 크기의 1/10만큼 움직인다. 구축자의 마지막 두행은 초기방향(오른쪽)과 초기위치(창문의 왼쪽측면으로부터 3cm, 창문의 꼭대기로부터 3cm)를 설정한다.

SlowBug::Move()의 사명은 벌레를 이동시키는것이다. 느린 벌레는 다음의 동작을 진행한다. 느린 벌레가 창문의 테두리에 부딪치면 반대방향으로 이동한다. 느린 벌레는 임의로 방향을 시간의 대략 10% 정도로 바꾼다.

SlowBug::Move()의 시작은 다음과 같다.

```
Erase();
//임의로 방향이 바뀐다.
if (Generate Percentage.Draw() < GetDirectionchangeProbability())
    ChangeDirection();
SetPosition(NewPosition());
Draw();
```

첫 단계에서는 현재 표시된 비트매프를 지워 버린다. 다음단계에서는 변화된 방향을 결정한다. GeneratePercentage객체를 사용하여 1과 100사이의 란수를 얻는다. 만일 란수가 DirectionChangeProbability보다 작다면 ChangeDirection()은 임의로 방향을 선택하여 벌레를 그 방

향으로 설정한다(ChangeDirection()성원함수는 목록 15-2에서 정의하였다). 그렇지 않으면 벌레는 현재방향을 그대로 유지한다. 일단 방향이 결정되고 새 위치가 계산되면 벌레를 그 위치에 다시 그린다.

Move()의 다음단계는 벌레가 창문의 테두리로 다가가는가를 결정한다. 그렇게 되면 벌레가 반대방향으로 가야 한다. 완성된 코드는 다음과 같다.

```
DirectionBugDirection = GetDirection();
float BugX = GetPosition().GetXDistance();
float BugXSize = GetBmp(GetDirection()).GetWidth();
float BugY = GetPosition().GetYDistance();
float BugYSize = GetBmp(GetDirection()).GetHeight();
// 주위를 돌 필요가 있을 때
if(BugDirection == Right && BugX +BugXSize + GetHorizMovement()
  >= GetWindow().GetWidth())
    SetDirection(Left);
else if (BugDirection == Left && BugX - GetHorizMovement() <= 0.0)
    SetDirection(Right);
else if (BugDirection == Down && BugY + BugYSize + GetVertMovement()
  >= GetWindow().GetHeight())
    SetDirection(Up);
else if (BugDirection == Up && BugY - GetVertMovement() <= 0.0)
    SetDirection(Down);
```

코드의 첫 블로크에서는 벌레에 대한 일부 필요한 정보를 얻는다. 벌레의 현재방향과 벌레의 x자리표, y자리표, 표시된 비트매프의 높이와 너비를 얻는다.

이 정보를 리용하여 벌레가 창문의 테두리에 부딪쳤는가를 결정한다. 실례로 벌레가 오른쪽으로 이동할 때 비트매프의 오른쪽테두리에 수평으로 움직일수 있는 량을 더한 값이 창문의 오른쪽 테두리를 지나면 벌레는 왼쪽으로 돌아선다(창문의 크기는 물론 모든 자리표들은 cm이다). 그림 15-6은 계산실례이다. 다른 방향들도 같게 계산한다.

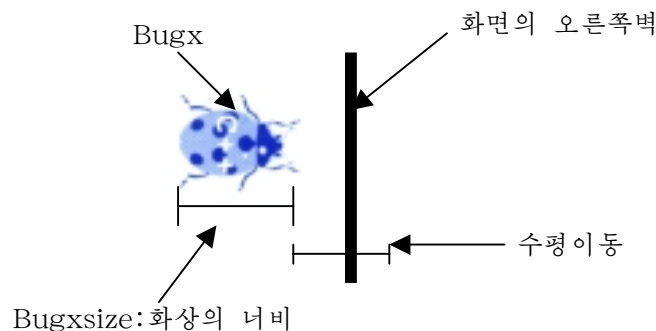



그림 15-6. 창문의 테두리로 접근하는 벌레

15.2.2 파생클라스 FastBug

Bug기초클라스를 사용하여 SlowBug와는 다른 동작을 수행하는 빠른 벌레를 창조하려고 한다. FastBug클라스선언은 SlowBug와 매우 류사하다.

```
class FastBug:public Bug {
    public:
        FastBug(SimpleWindow &w, int HitsNeeded=3,
          int DirectionChangeProbability=20);
        void Move();
};
```

FastBug는 SlowBug보다 작으며 3번 쳐야 죽는다. 그러나 SlowBug와 비교하여 보면 FastBug의 방향은 더 자주 변한다. FastBug구축자는 SlowBug구축자와 류사하다. 목록 15-4에 FastBug::FastBug()를 정의하였다.

FastBug구축자와 SlowBug구축자와의 차이는 적재된 화상모임이 다른것이고 HorizMovement와 VertMovement를 다르게 계산하며 초기방향과 위치를 다르게 설정한다는것이다. 그러나 기본 차이점은 FastBug에서 HorizMovement와 VertMovement가 더Slow Bug에 비해 크다는것이다.

목록 15-4. bug.cpp 에 있는 FastBug구축자

```
FastBug::FastBug(SimpleWindow &w, int h, int p ):Bug(w,h,p) {
    // 네개 방향으로 움직이는 벌레의 비트매프를 적재한다
    vector<string> BitMapFiles(BugBitMaps));
    BitMapFiles[0] = "fbug-u.bmp"
    BitMapFiles[1] = "fbug-d.bmp"
    BitMapFiles[2] = "fbug-l.bmp"
    BitMapFiles[3] = "fbug-r.bmp"
    for (Direction d = Up; d<=Right; d = (Direction)(d+1)) {
        GetBmp(d).SetWindow(GetWindow());
        GetBmp(d).Load(BitMapFiles[d]);
        assert(GetBmp(d).GetStatus() == BitMapOkay);
    }
    // 수직, 수평방향으로 벌레가 움직인 거리를 설정한다
    // 거리는 비트매프의 크기에 의존한다
    // 이 거리는 SlowBug보다 길어야 하므로 보다 빨리 움직여야 한다
    SetHorizMovement(GetBmp(Right).GetWidth() / 5.0);
    SetVertMovement(GetBmp(Up).GetHeight() / 5.0);
    // 초기방향을 아래로 설정
    SetDiretion(Down);
    SetPosition(Position(6.0, 2.0));
}
```

때문에 FastBug는 SlowBug보다 더 빨리 이동하는것처럼 보인다. FastBug::Move()는 Slow::Move()와는 아주 다르다. 물론 이것은 FastBug가 다르게 이동하여야 하기때문이다. 더 빨리

움직이며 창문벽을 뚫고 움직여야 한다. 일부 콤퓨터학자들은 HeisenBug라고 한다. 다만 FastBug와 SlowBug의 차이가 벌레들이 이동한 속도라면 새 클라스는 필요 없을것이다. 이 한가지 차이는 벌레의 (즉 고유한 비트매프를 적재하는것과 HorizMovement, VertMovement 를 적당하게 설정하는것) 일부 속성을 적당한 값으로 설정하여 처리되게 한다.

FastBug::Move()에서 첫 부분은 SlowBug와 같다. 코드는

```
void FastBug::Move() {
    Erase();
    // 우연히 방향을 바꾼다.
    if (Generate percentage. Draw() < GetDirection change Probability())
        ChangeDirection();
    SetPosition(NewPosition());
    Draw();
```

이다. 두번째 부분은 창문테두리에 부딪치는가를 결정한다. 창문의 테두리에 부딪칠 때 방향을 바꾸는 SlowBug와 달리 FastBug는 같은 방향을 유지하기때문에 창문의 테두리를 《뚫고》 들어 가는것처럼 보인다. 이 동작을 실현한 코드는 다음과 같다.

```
// 벌레의 위치와 크기를 얻기
Direction BugDirection = GetDirection();
float BugX = GetPosition().GetXDistance();
float BugXSize = GetBmp(GetDirection()).GetWidth();
float BugY = GetPosition().GetYDistance();
float BugYSize = GetBmp(GetDirecttion()).GetHeight();
// 벌레가 반대쪽에서 나오는가 결정
if (BugDirection == Right && BugX + BugXSize + GetHorizMovement()
  >= GetWindow().GetWidth()) {
    Erase();
    SetPosition(Position(1, GetPosition().GetYDistance()));
    Draw();
}
else if (BugDirection == Left && BugX - GetHorizMovement() <= 0.0) {
    Erase();
    SetPosition(Position(GetWindow().GetWidth - BugXSize,
      GetPosition().GetDistance()));
    Draw();
}
else if (BugDirection == Down && BugY + BugYSize >= GetWindow(0.GetHeight()) {
    Erase();
    SetPosition(Position(GEtPosition().GetXDistance(),0.0));
    Draw();
```

```
    }
    else if ( BugDirection == Up && BugY - GetVertMovement() <= 0.0) {
        Erase();
        SetPosition(Position(GetPosition().GetXDistance(),
          GetWindow().GetHeight() - BugYSize));
        Draw();
    }
```

벌레가 창문의 테두리에 있는가를 결정하는 론리는 SlowBug::Move()와 정확히 같다. 그 차이는 우연히 나타난다. FastBug에 대해서는 현재 화상을 지우고 화면의 반대쪽에 그린다. 방향은 같다.

SlowBug와 FastBug이동함수의 류사성은 벌레가 창문의 테두리에 있는가를 결정하는 코드를 단일화할수 있다는것을 암시한다. 이 함수들은 Bug기초클라스의 성원함수이기때문에 두 클라스에서 다 리용할수 있다. 다음 4개의 성원함수들 AtRightEdge(), AtLeftEdge(),AtBottomEdge()와 AtTopEdge()는 Bug클라스에서 보호부분으로 선언되였다. 이 함수들은 벌레가 대응하는 테두리에 있다면 참을 돌리고 아니면 거짓을 돌린다. 보호성원함수들은

```
    // 벌레가 창문의 오른쪽변에 있는가를 결정
    bool Bug::AtRightEdge() const {
        return (GetPosition().GetXDistance() + GetBmp(GeDirection()).GetWidth()
          + GetHorizMovement() >= GetWindow().GetWidth());
    }
    // 벌레가 창문의 왼쪽변에 있는가를 결정
    bool Bug::AtLeftEdge() const {
        return (GetPosition().GetXDistance() - GetHorizMovement() <= 0.0);
    }
    // 벌레가 창문의 아래쪽변에 있는가를 결정
    bool Bug::AtBottomEdge() const {
        return(GetPosition().GetYDistance() + GetBmp(GetDirection()).GetHeight()
          + GetVertMovement() >= GetWindow().GetHeight());
    }
    // 벌레가 창문의 웃쪽변에 있는가를 결정
    bool Bug::AtTopEdge() const {
        return(GetPosition().GetYDistance() - GetVertMovement() <= 0.0);
    }
```

와 같다. FastBug::Move()에서 이 함수들을 리용하여 벌레가 창문의 테두리를 뚫고 들어가는가를 결정하는 코드는

```
    if (BugDirection == Right && AtRightEdge()) {
        Erase();
        SetPosition(Position (0.0, GetPosition().GetYDistance()));
        Draw();
```

```
    }
    else if (BugDirection == Left && AtLeftEdge()) {
        Erase();
        SetPosition(
          Position(GetWindow().GetWidth() - BugXSize, GetPosition().GetYDistance()));
    Draw();
    }
    else if (BugDirection == Down && AtBottomEdge()) {
        Erase();
        SetPosition(Position(GetPosition().GetXDistance(), 0.0));
        Draw();
    }
    else if (BugDirection == Up && AtTopEdge()) {
        Erase();
        SetPosition(Position(GetPosition().GetXDistacnce(),
          GetWindow().GetHeight() - BugYSize));
        Draw();
    }
```

와 같다. SlowBug::Move()코드는 더 짧아 진다.

벌레를 정의하였으므로 이제부터는 유희조종자의 설계와 실현을 시작할수 있다.

15.3 GameController클라스

유희조종자의 사명은 유희를 조종하는것이다. 유희조종자는 유희의 상태를 유지하며 마우스찰칵사건과 박자발생사건을 처리하여야 한다. 유희조종자의 공개대면부에는 다음과 같은것들이 있다.

- Reset: 유희를 초기상태로 설정한다.
- Player: 유희가 시작하게 한다.
- MouseClick: 마우스찰칵사건을 처리한다.
- TimerTick: 박자발생사건을 처리한다.

이밖에 유희조종자에는 다음과 같은 공개자료성원들도 있다.

- GameWindow: SimpleWindow에 대한 지적자
- Level: 유희의 현재준위
- Status: 유희의 상태
- KindOfBug: 유희에서 사용된 여러가지 벌레형태들을 지적하는 벡토르

목록 15-5에 GameController클라스의 정의를 주었다. 코드에는 두개의 렬거형객체의 정의가 있다. 첫번째 렬거형객체는 유희의 준위를 정의한다. 이 유희에는 두개의 준위가 있다. 첫 준위는 느린 벌레잡기이고 두번째 준위는 빠른 벌레잡기이다. 두번째 렬거형객체는 각이한 배경자의 상태를 정의한다. 실례로 렬거형성원변수 SettingUp은 유희가 초기화되고 있으며 놀이를 시작할 준비가 되지 않았음을 지적한

다. 이외에 Playing은 유희가 진행중인가와 마우스찰칵사건과 박자발생사건을 적당히 처리하여야 한다는것을 지적한다. 상수 NumberOfBugType는 유희에 제공된 서로 다른 형태의 벌레들이 얼마인가를 정의한다. 설명문을 보는바와 같이 이 상수에는 유희의 준위수와 일치하여야 한다.

목록 15-5. bughunt.h에 있는 GameController클라스의 선언

```
#ifndef BWINDOW_H
#define BWINDOW_H
#include "bug.h"
enum GameLevel { Slow, Fast, Done };
enum GameStatus { GameWon, Playing, GameLost, SettingUp};
//유희의 속도
const int GameSpeed = 100;
class GameController {
    public:
        GameController(const string &Title = "Bug Hunt!",
          const Position &WinPosition = Position(3.0, 3.0),
          const float WindLength = 14.0,
          const float WinHeight = 10.0);
        ~GameController();
        SimpleWindow *GetWindow();
        void Reset();
        void Play(const GameLevel Level);
        int MouseClick(const Position &MousePosition);
        int TimerTick();
    private:
        void BugHit();
        GameLevel CurrentLevel() const;
        Bug *CurrentBug() const;
        SimpleWindow *GameWindow;
        GameLevel Level;
        GameStatus Status;
        vector<Bug*> KindOfBug;
};
#endif
```

GameController구축자는 창문과 벌레 등 모든 유희객체들을 할당한다. 이외에 자료성원 Level과 Status를 초기화한다. 이것을 실현한 코드는 아래와 같다.

```
GameController::GameController(const string &Title,
```

```
    const Position &WinPosition, float WinLength,
    float WinHeight) : Level(Slow), Status(SettingUp) {
      // 창문을 창조하고 열기
      GameWindow = new SimpleWindow(Title, WinLength, WinHeight, WinPosition);
      GetWindow() -> Open();
      // 벌레창조. 벌레는 반드시 창문이 열린 후에 창조되여야 한다.
      // 왜냐하면 벌레의 초기화는 창문에 의존하기때문이다.
      KindOfBug.reserve(NumberOfBugTypes);
      KindOfBug[Slow] = new SlowBug(*GetWindow());
      // 빠른 벌레창조
      KindOfBug[Fast] = new FastBug(*GetWindow());
    }
```

창문이 열린 후에 벌레들이 만들어 져야 한다는 설명이 코드에 들어 있다. 이것은 Bug가 구축될 때 그의 비트화상이 초기화되는데 이때 SimpleWindow형객체의 주소를 알고 있어야 하기때문이다.

GameController클라스는 동적으로 할당된 객체들을 해방하는 함수인 해체자를 가진다. 객체들에는 창문과 벌레들이 포함된다. 따라서 GameController::~GameController()는 이러한 객체들을 해방한다. 이것을 실현한 코드는 다음과 같다.

```
    GameController::~GameController() {
        // 벌레와 창문을 없앤다.
        delete KindOfBug[Slow];
        delete KindOfBug[Fast];
        delete GameWindow;
    }
```

공개성원함수 Reset()는 유희를 초기상태로 설정하는데 사용자가 벌레를 놓쳤을 때 사용한다. 유희는 처음부터 다시 시작된다. Reset()의 코드는 아래 와 같다.

```
    void GameController::Reset() {
        Status=Settingup;
        Level=slow;
        CurrentBug() -> Creat();
    }
```

Play()성원함수는 간단하다. 그것을 실현한 코드는 다음과 같다.

```
    void GameController :: Play (const GameLevel) {
        Level=1;
        Status=:Playing ;
        GetWindow()→StartTimer(GameSpeed);
    }
```

Play()는 단계를 설정하고 놀이의 상태를 변화시키며 현재 벌레가 움직일수 있도록 시간계수를 시

작한다. 유희조종은 대부분 마우스찰칵사건과 박자발생 사건을 처리하는데 돌려진다. 마우스를 누를때 유희가 진행되는가 ,벌레를 맞 혔는가를 검사하여야 한다. 벌레를 맞혔다면 성원함수 BugHit()를 호출하여 오락의 상태를 갱신한다. 유희에서 이겼는가를 보기 위하여 유희의 상태를 검사한다. 그러면 SimpleWindow알림창은 이김통보로 나타난다.

유희도중에 사용자가 벌레를 놓치면 시간계수기서고통보문이 표시되며 유희는 다시 시작된다. 목록 15-6에 GameController::MouseClick()의 코드를 주었다.

목록 15-6. control.cpp에 있는 MouseClick와 TimerTick함수

```
// 벌레가 눌리웠는가 검사하고 눌리웠으면 유희를 갱신한다.
int GameController::MouseClick(const Position & MousePosition) {
    // 유희가 진행중에 있다면 마우스에 주의를 준다
    if (Status == Playing && CurrentBug() ->IsHit(MousePosition)) {
        BugHit();
        // 유희에서 이겼다는것을 알려 준다
        if (Status == GameWon) {
            GetWindow() -> StopTimer();
            GetWindow() -> Message("You Won!");
            Reset();
            Play(Slow);
        }
    }
    else {
    // 벌레를 놓쳤을 때의 처리
        GetWindow() -> StopTimer();
        GetWindow() -> Message("You Missed!");
        CurrentBug() -> Kill();
        Reset();
        Play(Slow);
    }
    return 1;
}
//벌레를 움직인다.
int GameController:: TimerTick() {
    CurrentBug() ->Move();
    return 1;
}
```

GameController::TimeTick()코드를 목록 15-6에 주었다. 현재벌레에게 이동할것을 알리는 통보를 보낸다. C++가 지원하는 다형성을 리용하여 이 코드를 간단히 작성할수 있다.

이제는 벌레의 형태에 따라 적당한 Move()함수가 자동적으로 호출된다. 마지막으로 론하여야 할 GameController성원함수로서 비공개성원함수 BugHit()가 있다. 그것을 실현한 코드는 다음과 같다.

```
// 벌레를 눌렀다
void GameController::BugHit() {
    // 벌레가 죽으면 유희를 끝내고 아니면 유희를 계속한다
    if (CurrentBug() -> IsDying()) {
        CurrentBug() -> Kill();
        Level = (GameLevel) (Level + 1);
        If (Level == Done)
            Status = GameWon;
        else // 새로운 더 빠른 벌레를 창조
            CurrentBug() -> Create();
    }
}
```

코드는 현재벌레가 죽었는가를 결정한다. 그 벌레가 죽었으면 현재찰칵회수가 재설정되고 유희는 다음단계로 넘어 간다. 다음단계가 없다면 유희상태를 GameWon으로 설정한다. 그렇지 않으면 다음종류의 벌레가 창조된다.

15.4 벌레잡이유희프로그람

프로그람의 나머지부분은 유희를 창조하고 역호출을 설정하는 코드이다. 목록 15-7에 이 코드를 주었다.

목록 15-7. bughunt.cpp에서 역호출함수와 유희를 시작하고 끝내는 함수

```
#include "bughunt.h"
GameController *BugHunt;
//시간이 될 때마다 호출되는 함수
int TimerCallback(void) {
    BugHunt -> TimerTick();
    return 1;
}
// 마우스가 눌리울 때마다 호출되는 함수
int MouseCallback(const Position &MousePosition) {
    BugHunt -> MouseClick(MousePosition);
    return 1;
}
//유희조종기를 할당하고 역호출을 설정한 다음 유희를 시작한다
int Apimain() {
```

```
    EzRandomize();
    BugHunt = new GameController();
    (BugHunt -> GetWindow()) ->
      SetTimerCallback(TimerCallback);
    (BugHunt ->GetWindow()) ->
      SetMouseClickCallback(MouseCallback);
    BugHunt -> Play(Slow);
    return 0;
}
// 유희를 끝낸다.
int ApiEnd() {
    delete BugHunt;
    return 0;
}
```

ApiMain()함수는 EzWindow모조란수발생기를 초기화한 다음 Bug Hunt라는 유희를 창조한다. 마우스와 시계역호출을 다 설정하고 Slow준위에서 유희가 시작된다. ApiEnd()는 끝내기통보문을 보낼 때 호출되며 유희를 삭제한다. 이 기능은 물론 유희조종자의 해체자를 호출하여 벌레들과 창문을 삭제한다.

이것으로 Bug Hunt의 설계와 실현을 완성한다. 이 장에서 처음에 언급한것와 같이 Bug Hunt는 3개의 모듈(bug.cpp 와 control.cpp, bughunt.cpp)들의 대면부(bug.h, control.h, bughunt.h)와 EzWindows API서고들로 이루어 져 있다. CD-ROM에 완성된 코드가 있다. Bug Hunt의 설계는 많은 유희설계의 본보기로 된다. 조종자는 유희놀이를 조종하고 유희의 상태를 보존한다. 시작코드에 마우스찰칵를 처리하기 위한 프로그람들을 묶에서 사용자와 마우스찰칵사건사이의 대면부를 설정한다. 이 모형은 재미난 유희를 개발하는데 리용될수 있다. 일부는 련습에서 보기로 하자.

그림 15-7. 15조각맞추기유희의 화면

그림 15-8. 전자수산기응용프로그람의 화면

문 제

1. Bug클라스는 벌레의 비트매프를 가진 벡토르를 사용한다. 비트매프의 배렬을 사용할수 있도록 Bug클라스를 수정하시오. 수정한 코드와 원래코드를 비교하시오. 어떤 방법이 좋고 어떤 방법이 나쁜가?
2. 15조각맞추기유희를 작성하시오. 초기에 유희는 15개의 번호가 달린 4각형과 빈 4각형을 4×4형으로 묶은 4각형으로

구성된다. 어떤 사람이 이것을 규칙없이 조합하면 그것들을 다시 이동시켜 순서대로 조합하여야 하였다. 15쪼각맞추기가 1865년에 제기되였을 때 매 쪼각마다 번호를 달아 주었다. 현재 15쪼각맞추기를 많이 하는데 번호대신 그림을 사용한다. 이 문제에서도 그림을 사용한다. 그림 15-7에 이 유희의 표본화면을 주었다.

3. 전자수산기는 대체로 조작체계에 포함되여 제공된다. 전자수산기응용프로그람을 작성해 보시오. 이런 응용프로그람들은 우리가 쓰는 전자수산기를 모형화한 사용자대면부를 제공한다. 그림 15-8에 전자수산기응용프로그람의 화면을 주었다.

15.5 알아 둘 점

✓ 계승은 코드를 재리용할수 있는 방법을 제공한다. Bug Hunt에서 Bug기초클라스의 코드는 파생클라스 SlowBug와 FastBug에 의하여 사용된다. 새로운 벌레가 유희에 추가된다면 기초클라스를 리용할수 있다.

✓ 포함은 다른 객체들로 복합객체를 만드는 방법이다. Bug Hunt에서 유희조종클라스는 포함을 리용하여 작성된다. 유희에는 창문(즉 SimpleWindow)과 여러가지 종류의 벌레들(즉 KindOfBug묶음)이 포함된다.

✓ 교갑화는 객체의 상세한 내용을 사용자들로부터 은페시키는 쏘프트웨어를 작성하기 위한 방법이다. Bug Hunt에서 유희조종자는 Bug클라스를 리용한다. 벌레가 어떻게 표현되며 어떻게 움직이는가는 Bug클라스에 은페시켰다. 조종자는 벌레가 어떻게 표현되고 움직이는지 모르지만 Bug의 공개성원함수 Move()를 호출하여 이동시킬수 있다.

✓ 다형성은 객체의 종류에 관계없이 계승과 관련된 객체를 조작하는 코드를 작성할수 있게 한다. Bug Hunt안에서 유희조종자는 FastBug인가 SlowBug인가에 관계없이 벌레를 조작할수 있다. 례를 들어 Move()성원함수는 다형성함수이다. 다형성을 가진것으로 하여 유희조종자는 벌레가 이동하도록 지시할 때 벌레의 형에 맞는 Move()함수를 호출한다.

콤퓨터의 력사

병렬계산과 제기되는 문제

콤퓨터의 기본계산은 직렬계산방식이다. 매 계산은 순서대로 다음동작을 시작하기전에 계산을 완성하면서 진행된다. 병렬계산이라는것은 많은 계산량들을 한번에 병렬적으로 처리한다는것이다. 물론 이러한 동작을 수행하려면 여러개의 콤퓨터소편이 필요하다. 한 사람이 작업하는것보다 다섯명의 사람이 작업을 하면 더 빠른것처럼 많은 콤퓨터가 문제 하나를 푸는것은 하나의 콤퓨터로 할 때보다 더 빠르다.

물론 병렬계산에서는 문제를 푸는 콤퓨터의 수가 매우 커지면 진행한 작업을 구분하고 지적하기가 힘들다. 만일 100명의 사람들이 집청소를 도와 준다면 그때 생기게 되는 복잡성을 생각해 보시오.

새로운 기술의 도입으로 아무리 어려운 문제도 쉽게 풀릴수 있게 되였다. 연구사들은 병렬계산에서 큰 전진을 이룩하였다. 그들은 병렬체계를 개발하는데 기본력량을 집중하였으며 이 문제를 여러가지로 고찰하였는데 이러한 문제를 《큰 도전》(Grand Challenge)이라고 한다. 병렬계산은 과학과 기술에서 기본 해결해야 할 문제이다. 이 중요한 문제를 해결하면 높은 기능의 계산기술과 자원을 개발하는데서 큰 전진을 이룩할수 있다. 이 문제들에는 복잡한 문제를 모형화하기 위한 콤퓨들의 리용이 포함된다.

병렬계산을 리용하여 날씨의 정확한 예보와 지진을 예측할수 있는 정보같은것을 쉽게 구할수 있다. 또

한 자연환경이 생태계에 주는 영향도 빨리 구할수 있다.

《큰 도전》은 가상현실이라는 문제를 가지고 있다. 가상현실(Virtual Reality)은 현실과 구별되는 인공적인 환경을 창조하기 위해 콤퓨터를 리용하고 있다. 원시적인 가상현실체계는 이미 리용되고 있다. 대부분의 비행사들은 모의기를 통하여 새로운 비행기조종법을 배운다. 모의기는 매우 생동하며 안전하고 값도 눅다. 가상현실체계는 콤퓨터도형학에 의하여 더욱 완성될것이다.

련습문제

15.1. Bug Hunt에서 2개의 렬거형객체들인 GameLevel과 GameStatus는 control.h에서 정의된다. 이것은 GameController클라스에 실지 교갑화되여야 한다. GameController클라스에 교갑화시키기 위하여 벌레잡이유희의 코드를 수정하시오. 코드가 변경된 매 모듈의 시작위치에 설명문을 달아주시오.

15.2. Bug Hunt프로그람을 수정하여 WarpBug라는 새로운 형의 벌레를 정의하시오. 이 벌레는 공간상에서 움직이는데 일정한 간격마다 사라졌다가 창문의 임의의 위치로 돌아 간다. WarpBug벌레를 잡으려면 벌레를 두번 쳐야 한다. 즉 새로운 Bug Hunt유희는 세 준위로 구성된다.

15.3. Bug Hunt프로그람에서 불충분한것은 사용자가 마우스로 벌레를 잡은 경우 점수가 없다는것이다. 사용자가 벌레를 잡았을 때 점수를 주는 형식을 생각해 보시오. Bug Hunt프로그람을 수정하여 이것을 실현하시오.

15.4. 거의 모든 벌레들은 한번 맞으면 더 빨리 움직인다. Bug Hunt프로그람을 수정하여 벌레가 한번 맞으면 보다 빨리 움직이도록 하시오.

15.5. 벌레를 놓치는 경우 다른 벌레가 나타나도록 Bug Hunt프로그람을 수정하시오. 창문안에서 기여다니는 벌레들의 수는 3을 초과하지 말아야 한다. 창문에 벌레들이 여러마리 있을 때의 현상을 관찰하시오. 이런 현상이 왜 생기는가?

15.6. 시계를 리용하여 설정시간내에 모든 벌레를 잡지 못하면 유희를 끝내도록 Bug Hunt프로그람을 수정하시오.

15.7. 시계를 리용하여 벌레를 모두 잡았을 때의 시간(초단위로)을 기록하도록 Bug Hunt프로그람을 수정하시오. 파일에 점수를 기억시키고 유희의 마지막에 그것들을 표시하시오.

15.8. Balloons라고 하는 유희를 새로 설계하고 실현하시오. 이 유희에서 고무풍선은 창문의 바닥에서 나타나 꼭대기로 날아 난다. 사용자는 풍선이 창문의 꼭대기에 도달하기전에 풍선을 《뽑아》야 한다. 풍선이 꼭대기에 이르면 사용자는 유희에서 패하게 되며 유희가 다시 시작된다. 풍선들은 각이하게 움직인다. 매 수준은 그전의 수준보다 풍선을 뽑기가 더 힘들다.

15.9 Terminator라고 하는 유희를 새로 설계하고 작성하시오. 이 유희에서 사용자는 로보트를 조종하게 된다(10장에 있는 비트매프를 사용하시오). 유희의 목적은 로보트로 벌레들을 추적하여 《죽이는》 것이다. 앞에서 본 Bug Hunt유희에서와 마찬가지로 느린 벌레와 빠른 벌레가 있다. 로보트의 운동을 조종하기 위하여 조종화살표를 가진 매개의 창문을 창조하여야 한다. 화살표를 눌러 로보트를 그 방향으로 한 걸음 이동시킨다. 유희를 재미 있게 하기 위하여 벌레가 벽을 뚫고 이동할수 있도록 설계할수 있다. 즉 벌레가 창문의 벽에 부딪칠 때 반대위치에 나타나야 한다. 이때 로보트는 벽을 통과할수 없다. 종류가 다른 벌레들을 늘일수 있게 또한 벌레들의 마리수를 쉽게 늘일수 있게 유희를 설계하시오. 처음에는 느린 벌레 3마리와 빠른 벌레 2마리가 움직이는 유희를 작성하시오.

부록 1. ASCII문자표와 C++의 연산자우선권표

1.1 ASCII문자표

코 드	문 자	이 름	코 드	이 름	코 드	이 름	코 드	이 름
0	^@	NUL	32	SP	64	@	96	‘
1	^A	SOH	33	!	65	A	97	a
2	^B	STX	34	"	66	B	98	b
3	^C	ETX	35	#	67	C	99	c
4	^D	^DOT	36	$	68	D	100	d
5	^E	ENQ	37	%	69	E	101	e
6	^F	QCK	38	&	70	F	102	f
7	^G	BEL	39	’	71	G	103	g
8	^H	BS	40	(	72	H	104	h
9	^I	TAB	41	)	73	I	105	I
10	^J	LF	42	*	74	J	106	j
11	^K	VT	43	+	75	K	107	k
12	^L	FF	44	,	76	L	108	l
13	^M	CR	45	-	77	M	109	m
14	^N	SO	46	.	78	N	110	n
15	^O	SI	47	/	79	O	111	o
16	^P	DEL	48	0	80	P	112	p
17	^Q	DC1	49	1	81	Q	113	q
18	^R	DC2	50	2	82	R	114	r
19	^S	DC3	51	3	83	S	115	s
20	^T	DC4	52	4	84	T	116	t
21	^U	NAX	53	5	85	U	117	u
22	^V	SYN	54	6	86	V	118	v
23	^W	ETB	55	7	87	W	119	w
24	^X	CAN	56	8	88	X	120	x
25	^Y	EM	57	9	89	Y	121	y
26	^Z	SUB	58	:	90	Z	122	z
27	^[	ESC	59	;	91	[	123	{
28	^¥	FS	60	<	92	¥	124	|
29	^]	GS	61	=	93	]	125	}
30	^^	RS	62	>	94	^	126	~
31	^_	US	63	?	95	_	127	DEL

1.2 연산자우선권

다음의 표는 C++연산자의 우선권을 보여 준다. 같은 우선권을 가진 연산자는 같은 행에서 서로 분리하였다.

연산자	기 능	종 류	놓이는 위치
()	함수호출	앞붙이	왼쪽
[]	첨수	앞붙이	왼쪽
. ->	선택, 간접선택	앞붙이	왼쪽
::	유효범위해결연산자	앞붙이	왼쪽
++ --	증가, 감소	앞붙이	오른쪽
! ~	론리부정, 비트부정	단항연산자	오른쪽
+ -	산수더하기, 덜기	단항연산자	오른쪽
++ --	증가, 감소	앞붙이	오른쪽
& *	주소, 간접	단항연산자	오른쪽
sizeof	기억기크기	단항연산자	오른쪽
()	형변환	단항연산자	오른쪽
new delete	기억기할당, 해제	단항연산자	오른쪽
.*	성원선택	앞붙이	왼쪽
->*	성원지적자선택	앞붙이	왼쪽
* / %	곱하기, 나누기, 나머지연산	2항연산자	왼쪽
+ -	산수더하기, 덜기	2항연산자	왼쪽
<< >>	밀기	2항연산자	왼쪽
< <= > >=	비교연산	2항연산자	왼쪽
== !=	같기, 같지않기	2항연산자	왼쪽
&	비트곱하기	2항연산자	왼쪽
^	비트배타론리	2항연산자	왼쪽
|	비트더하기론리	2항연산자	왼쪽
&&	론리곱하기	2항연산자	왼쪽
||	론리더하기	2항연산자	왼쪽
? :	조건식	3항연산자	오른쪽
= *= /= %= += -= <<= >>= &= ^= |=	대입	2항연산자	오른쪽
throw	례외	단항연산자	오른쪽
,	순차평가	2항연산자	왼쪽

부록 2. 표준서고

C++표준은 확장된 서고를 결합하고 있다. 일부 서고들은 assert, ctype, error, float, locale, math, setjmp signal, stdarg, stddef, stdio, stdlib, string, time과 같이 원래의 C서고들이다. 다른 서고들은 C++에서만 고유한것으로서 algorithm, deque, exception, fstream, iomanip, iostream, iterator, limits, list, locale, map, new, numerics, queue, set, stack, string, stringstream, utility, vector들이다. 이 서고들을 명백히 리해하기 위해서는 참고서, 콤파일러문서, 혹은 다음의 서적들중의 하나를 참고하시오.

- B.Stroustrup, The C++ Programming Language, 3rd, Reading, MA:Addison-Wesley, 1998.
- International Standard for Information Systems-Programming Language C++, ISO/IEC FDIS 14882, Washington, DC:American National Standards Institute, 1998.
- P.J.Plauger, The Standard C Library, Englewood Cliffs, NJ:Prentice-Hall, 1992
- P.J.Plauger(editor), A.A.Stepanov, M.Lee, and D.R>Musser, The C++ Standard template Library, Englewood Cliffs, NJ:Prentice-all, 2000.

2.1 서고이름과 접근

표준적인 C++에서 표준서고에 대한 접근은 머리부뒤붙이를 사용하지 않는다. 실례로 다음의 명령문은 iostream서고를 포함한다.

```
#include <iostream>
```

이와 함께 표준서고를 얻는 C에 대한 접근에서도 머리부뒤붙이를 사용하지 않는다. 그러나 C서고이름은 그 원천을 가리키도록 앞에 c를 붙여야 한다. 실례로 다음의 명령문은 assert마크로서고를 포함한다.

```
#include <cassert>
```

대부분의 콤파일러들은 서고를 머리부뒤붙이를 가지거나 뒤붙이가 없이 포함하도록 하여야 한다. 실지로 여러가지 콤파일러들은 이 조항의 여러개의 머리부뒤붙이를 사용한다. 그러한 콤파일러들에 대하여 보면 iostream과 assert서고들은 다음과 같이 접근될수 있다.

```
#include <iostream.h>
#include <assert.h>
```

머리부뒤붙이를 리용하는 판본은 대체로 전역령역에 선언하며 뒤붙이를 쓰지 않는 판본은 std이름공간에 선언한다(이름공간에 대하여서는 부록 4에서 본다).

다음부분에서 론의는 일부 C와 C++서고들에서 제한된 함수들, 형들, 마크로들과 객체들의 선택을 보여 준다. 다른 중요한 표준서고들은 본문들에서 소개되였다(즉 3장은 문자렬서고를 소개하였다. 5장은 C형, assert, iomanip서고들에 대하여 론의하였다. 9장에서는 표준본보기서고로부터 벡토르와 다른 용기클라스들을 고려하였다). 추가적으로 부록 3에서는 문자렬클라스와 일부 다른 용기클라스들에 대하여 구체적으로 준다.

2.2 iostream서고

iostream서고와 계층도는 본문에서 론의되고 사용된다. 삽입 및 추출연산자와 조작자 그리고 전역흐름객체 cin, cout, cerr, clog들을 정의하는것외에 서고는 ios, istream, ostream성원함수에 대한 접근을 제공한다. istream성원함수에는 다음과 같은것들이 있다.

int istream::get()

입력흐름이 추가적인 자료를 포함한다면 그 함수는 다음문자를 추출하고 돌려 준다.만일 그렇지 않으면 EOF를 돌려 준다.

istream & istream :: get(**char** &c)

만일 입력흐름이 추가적인 자료를 포함한다면 그 함수는 C에서 다음문자를 추출하고 할당한다. 그렇지 않으면 C에서 결과가 정의되지 않는다. *this(요구하는 객체)에 대한 참조를 돌려 준다.

istream& istream :: get(**char** s[], **int** n;**char** delim=' \n')

입력흐름으로부터 문자를 추출하고 다음조건이 발생할 때까지 s에 이 문자들을 할당한다. 즉 n-1개의 문자가 추출되였다면 더이상 추출할 문자가 없으므로 추출되는 다음문자는 delim값을 가진다. 만일 마지막조건이 발생한다면 구분기호는 추출되지 않는다. 빈 완료문자는 s에 복사된 마지막추출값다음에 놓인다. *this에 대한 참조를 돌려 준다.

istream& istream :: getline(**char** s[], **int** n;**char** delim=' \n')

입력흐름으로부터 문자를 추출하고 다음조건이 발생할 때까지 s에 이 문자들을 할당한다. 즉 n-1개의 문자가 추출되고 더이상 추출한 문자가 없으면 추출하는 다음문자는 구분기호이다. 만일 마지막조건이 발생하면 구분기호는 추출되지만 s에 할당되지 않는다. 빈 완료문자는 s에 복사된 마지막추출값다음에 놓인다. *this 의 참조를 돌려 준다.

int stream :: peek()

입력흐름이 추가적인 자료를 포함한다면 함수는 추출된 다음문자를 돌려 준다. 그렇지 않으면 함수는 EOF를 돌려 준다.

istream& istream :: unget(**char** c)

문자 C가 입력흐름에 들어 간것이다. 문자 C는 추출되는 다음문자로 된다. *this에 대한 참조를 돌려 준다.

iostream서고는 일부 프로그람작성자들이 파일의 끝을 검색하는데 리용하는 ios성원함수에 대한 접근을 제공한다.

bool ios :: eof()

파일의 끝이 흐름에 이르렀다면 참을 돌린다. 그렇지 않으면 함수는 거짓을 돌린다.

iostream서고는 역시 get()와 getline()과 류사한 두 출력흐름성원함수들을 제공한다.

ostream& ostream :: put(**char** c)

출력흐름에 c문자를 삽입한다. *this에 대한 참조를 돌려 준다.

ostream& ostream :: write(**const char** s[], **int** n)

s로부터 출력흐름에 n개의 문자들을 삽입한다. 빈 문자도 있을수 있다. *this의 참조를 돌

려 준다. iostream서고는 또 다른 ostream성원함수를 제공한다.

ostream& ostream :: flush()
끝내기 위하여 아직 끝나지 못한 삽입연산자를 요구한다. *this참조를 돌려 준다.

2.3 stdlib서고

C의 stdlib서고는 여러가지의 형들, 함수들 그리고 마크로정의들의 집합이다. 서고에서 선언된 기본형들은 옹근수형 size이다. 서고에서 가장 많이 리용되는 함수들은 다음과 같다.

int abs(**int** n)
n의 절대값을 돌린다.

double atof(**const char** s[])
문자렬 s에 의해 표시된 수에 대하여 **double**형을 돌려 준다. 만일 문자렬이 수자로 표시되지 않으면 함수의 값은 정의되지 않는다.

long int atoll(**const char** s[])
문자렬 s로 표시된 수에 대하여 **long int**형을 돌려 준다. 문자렬이 수자로 표시되지 않으면 함수의 값은 정의되지 않는다.

void exit(**int** status)
프로그람은 status의 되돌림값을 가지고 끝난다.

void free(**void** *ptr)
빈 기억기에서 ptr가 지적하는 곳의 기억기를 준다.

void *malloc(size_t size)
현재 할당된 빈 기억기의 byte크기로 지적자를 돌려 준다. 만일 충분한 기억공간이 없다면 0을 돌린다.

int rand()
RAND_MAX의 범위내에서 모조란수를 돌려 준다. 순서발생을 위한 기정값은 1이다.

void srand(unsigned **int** val)
모조란수발생기에 의하여 나타난 val을 설정한다.

int system(**const char** s[])
지령처리에서 실행하는 조작체계로 문자렬 s를 넘긴다. 체계의존옹근수값을 돌려 준다.

2.4 math서고

쏘프트웨어개발에서 제일 품이 드는것은 과학적인 프로그람을 작성하는것이다. 과학적으로 프로그람을 작성하는것은 수학적인 공식과 모형을 리용하여 모든 대상을 확장시킨다는것을 의미한다. 이와 같은 프로그람들은 자주 삼각법과 제곱 및 로그함수들을 사용한다. 쏘프트웨어를 쉽게 개발하기 위하여 C의 math서고는 많은 math함수들을 제공하고 있다.

부록에서 론의된 다른 표준서고와는 달리 math함수의 정의는 프로그람이 번역될 때 반드시 자동련결되지 않는다. 지령행으로부터 호출되는 콤파일러들에 대하여 련결은 파라메터를 통하여 콤파일지령에 쓰인다.

규모가 큰 프로그람작성환경부분인 콤파일러에 의하여 련결은 적당한 서고선택을 진행하는것으로 요구한다. math서고는 환경에 따라 다르게 취급되며 프로그람작성자는 다른 서고에 련결하여 실행할수도 있다. math서고에서 자주 리용되는 함수들은 다음과 같다.

acos(x)
    코시누스값이 x인 각도를 돌려 준다.

asin(x)
    시누스값이 x인 각도를 돌려 준다.

atan(x)
    탕겐스값이 x인 각도를 돌려 준다.

ceil(x)
    각 x의 코시누스값을 돌려 준다.

cosh(x)
    각 x의 쌍곡시누스값을 돌려 준다.

exp(x)
    e의 x제곱을 돌려 준다.

fabs(x)
    x의 절대값을 돌려 준다.

floor(x)
    x와 같거나 그보다 더 작은 옹근수들중에서 가장 큰 값을 돌려 준다.

log(x)
    x의 자연로그값을 돌려 준다.

log10(x)
    x의 상용로그값을 돌려 준다.

pow(x, y)
    x의 y제곱을 돌려 준다.

sin(x)
    각 x의 시누스값을 돌려 준다.

sinh(x)
    각 x의 쌍곡시누스값을 돌려 준다.

sqrt(x)
    각 x의 2차뿌리를 돌려 준다.

tan(x)
각 x의 탕겐스값을 돌려 준다.

tanh(x)
각 x의 쌍곡탕겐스값을 돌려 준다.

2.5 time서고

C의 time서고는 달력과 프로그람시간을 처리하기 위한 형, 함수, 마크로정의들의 집합체이다. 이 서고에 선언된 기본형은 clock_t, time_t와 tm이다. clock_t와 time_t형들은 정수형이다. tm형은 다음과 같은 struct형이다.

```
struct tm {
int tm_see     //seconds after the current minute
int tm_min     //minutes after the current hour
int tm_mday   //day of month
int tm_mon    //month of year
int tm_year   //year since 1900
int tm_wday //days since Sunday
int tm_yday   //days since January
int tm_isdst    //flag indicating if day light saving time is in effect:
               //positive value means in effect, zero value means
               //not in effect, negative value means unknow
```

이 서고에서 함수들은 현재날자와 그레고리달력을 사용한 날자인 달력시간과 특수한 시간지역에 의존하는 실행을 위한 달력시간인 국부시간을 취급한다. 이 서고에 포함된 일부 함수들은 다음과 같다.

char *asctime(**const** tm *tptr)
tm객체 *tptr의 표현인 문자렬에 대한 지적자를 돌려 준다.

clock_tclock()
clocks_per_sec로 구분될 때 프로그람이 리용되는 처리시간의 초수에 근사하는 clock_z값을 돌려 준다.

double difftime(time_t tl, time_t t2)
시간 t1과 t2사이의 **double**값을 돌린다.

tm* localtime(**const** time_t *tptr)
*ptr로 표시되는 달력시간의 국부시간 tm에 대한 지적자를 돌려 준다.

time_t mktime(tm *tptr)
객체 *tptr로 표시된 국부시간의 달력시간에 대한 time_t를 돌려 준다. 객체 *tptr는 역시 수정된다.

time_t time(time_t *tptr)
현재달력시간에 대한 time_t를 돌려 준다. tptr가 빈 문자가 아니라면 *tptr는 현재달력시

간으로 설정한다.

2.6 cstring서고

C의 cstring서고에는 문자렬을 조종하기 위한 함수들이 있다. NULL마크로도 정의한다. 이 서고에 정의된 일부 함수들은 다음과 같다.

char *strcat(**char** *t, **char** *s)

문자렬 s의 마지막에 있는 빈 문자까지 포함하여 문자렬 s의 복사를 진행한다. 복사는 z가 끝나는 빈 문자를 포함하는것으로 시작한다. 함수는 z를 돌려 준다.

char *strchar (**const char** *t, **int** c)

문자 c가 문자렬 z에 없다면 이 함수는 NULL을 돌려 준다. 그렇지 않으면 함수는 문자렬 z에서 처음으로 나타난 문자 c에 대한 지적자를 돌려 준다.

int strcmp(**char** *t, **char** *s)

문자렬 t와 s를 순서대로 비교한다. 두 문자렬들이 같다면 함수는 0을 돌려 준다. 만일 s가 t보다 앞에 놓이는 문자라면 함수는 부수값을 돌려 준다. 그렇지 않으면 함수는 정수값을 돌려 준다.

int strcmp(**char** *t, **char** *s, size_t n)

문자렬 s와 t에 대하여 첫 n개의 문자만을 비교한다. 함수의 되돌림결과는 우의 함수와 같다.

char *strcpy(**char** *t, **char** *s)

문자렬 s를 문자렬 t에 빈 문자를 포함하여 복사한다. 함수는 t를 돌려 준다.

size_t strlen(**const char** *s)

문자렬 s의 길이를 구하여 그 값을 돌려 준다(빈 문자는 제외하고 계산한다).

char *strncat(**char** *t, **char** *s, size_z)

문자렬 t의 마지막으로부터 첫 n개의 문자렬을 복사한다. 문자렬 s의 길이가 n보다 작다면 그 나머지자리에 빈 문자를 추가한다. 함수는 t를 돌려 준다.

char *strncpy(**char** *t, **char** *s, size_t)

문자렬 t에 문자렬 s의 첫 n개 문자를 복사한다. s의 길이가 n보다 더 작다면 그뒤에 빈 문자가 추가된다. 함수는 t를 돌려 준다.

char *strrchar(**const char** *t, **int** c)

문자 c가 문자렬 t에 없다면 함수는 NULL을 돌려 준다. 그렇지 않으면 함수는 문자렬 t에서 마지막으로 나타나는 문자 c의 지적자를 돌려 준다.

char *strstr(**const** *t, **const char** *s)

문자렬 s의 길이가 0이라면 함수는 t를 돌려 준다. 문자렬 s가 문자렬 t의 한 부분이라면 함수는 NULL을 돌려 준다. 그렇지 않다면 문자렬 t에서 문자렬 s가 처음 나타나는 위치에 대한 지적자를 돌려 준다.

2.7 알고리듬서고

표준본보기서고는 배렬, 용기, 객체를 처리하기 위한 50개이상의 방법을 가지고 알고리듬서고를 제공한다. 이러한 방법들은 머리부파일<algorithm>에 정의되여 있다. 여기서 이러한 방법들의 일부를 서술한다.

이 방법을 사용하는데서 여러 조건은 반복자, 렬 그리고 사용된 귀환형에서 처리된다. 그 조건은 다음과 같다.

- 다른형 diff_type가 필수적이다.
- 반복자증가연산자 ++는 입력, 출력, 쌍방향반복자에 대해서도 다 적용할수 있다. 만일 렬에서 다음요소가 없다면 반복자는 보호요소를 가리킨다.
- 입력, 출력반복자는 한번만 넘겨 져야 한다(입력, 출력반복자를 리용하는 알고리듬은 단순한 넘기기로 된다).
- 감소연산자는 순환에서 앞의 요소를 가리키게 한다. 만일 다른 요소가 없다면 결과는 반드시 정의되지 않는다.
- 첨자연산자 《 》는 임의접근반복자를 정의하는데 필요하다. 만일 a가 임의접근반복자이고 i가 옹근수라면 a[i]는 a의 시작점에서 i번째 요소의 참조로 된다.
- 론리값을 돌려 주는 함수도 있다. 이 함수는 참조해제되는 반복의 파라메터를 가진다.
- 렬은 시작과 끝반복자 s와 e에 의해 정의된다. 시작반복자 s는 순환의 첫 요소를 지적한다. 끝반복자 e는 순환의 마지막요소를 지적한다.

중요한 알고리듬서고본보기함수는 for_each()와 sort()이다. 이 함수들은 다음과 같이 지정한다.

```
template<class Iter, class Func>
  Func for_each(Iter s, Iters, func f)
```

입력반복자 s와 c로 정의된 렬에서 매 요소에 대한 함수 f를 적용한다. 파라메터 f를 귀환값으로서 사용한다.

```
template<class Iter>
  void sort(Iter s, Iter e)
```

임의접근반복자 s와 e로 정의된 요소의 렬을 분류한다.

다음의 프로그람에서 이 함수들을 리용하였다.

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
#include "randint.h"
void set (int &Val) {
```

```
        RandomInt u(1,100);
        val=u.Draw( );
    }
    void Display(int val){
        cout <<"  " <<val;
    }
    int main() {
        EzRandomize( );
        vector<int> A(5);
        for_each(A.begin(),A.end(),set);
        for_each(A.begin(),A.end(),Display);
        cout<<endl;
        sort(A.begin(),A.end());
        for_each(A.begin(),A.end(),Display);
        cout<<endl;
        return 0;
    }
```

알고리듬서고함수들 for_each와 sort()를 사용하는것외에 프로그람은 그 부분에서 정의되는 함수 set()와 Display()를 사용한다.

main()함수는 독립적인 값에 대한 모조란수발생기를 설정하기 위하여 EzRandomize()를 먼저 호출한다. 그다음 main()함수는 5개의 옹근수요소로 된 벡토르 A를 정의한다(매 요소는 기정값 0으로 초기화된다). for_each()함수는 그다음 호출된다. 처음 for_each()함수에서 함수 set()는 A의 매 요소를 처리하는 함수파라메터를 가지고 호출된다. 처음 for_each()함수호출에서 함수 set()는 A의 매 요소에 대하여 반복하여 호출한다. 함수 set()는 1부터 100사이의 모조란수값을 요소에 대입하여 주어 진 요소를 수정한다. for_each()함수는 함수 Display()를 리용하여 A의 요소를 반복현시하기 위해 두번째로 호출된다. 요소들은 함수 sort()에 의해 재배치된다. for_each()함수는 요소들을 다시 현시하기 위해 세번째로 호출된다. 프로그람의 실행결과는 다음과 같다.

```
31 83 91 57
47 57 83 91
```

서고에 의해 제공된 다른 기본알고리듬도 있다.

```
template<class Iter, class T>
  bool binary_search(Iter s, Iter e, const T&V)
```

반복자 s와 c의 앞에서 정의된 순차분류값들은 값 V에 의해 검사된다. V가 순서대로 놓여 있다면 참을 돌려 준다. 그렇지 않으면 거짓을 돌려 준다.

template <**class** In, **class** Out>

Iter copy(In s1,In c1,Out s2)

입력반복자 s1과 c1이 정의된 렬의 값은 출력반복자 s2가 지적하는 위치에서 시작하여 순차적으로 복사된다. 함수의 귀환값은 새로운 복사의 표기에 대한 반복자이다.

template <**class** Iter,**class** T>

Diff_type count (Iter s,Iter s,**const** T &V)

입력반복자 e에 의해 정의된 렬에서 값 v의 출현회수를 돌려 준다.

template <**class** Iter,**class** Pred>

Diff_type count_if (Iter s,Iter s,pred p)

입력반복자 s와 e 에 의하여 정의된 렬에서 p가 참일 때 출현회수를 계산한다.

template <**class** Iter ,**class** Pred>

bool equal(Iter s1,Iter e1,Iter s2)

입력반복자 s1과 e1에 의하여 정의된 렬의 요소가 s2에서 시작한 렬에서 해당한 요소와 같다면 참을 돌려 준다. 다른 경우 거짓을 돌려 준다.

template <**class** Iter,**class** T>

Iter find(Iter s,Iter s ,**const** T &V)

입력반복자 s와 e에 의하여 정의된 렬에서 값 v의 첫 출현에 대한 반복자를 돌려 준다. 렬에서 v가 생기지 않으면 e를 돌려 준다.

template <**class** Iter,**class** pred>

Iter find_if(Iter s,Iter s,pred p)

입력반복자 s와 e에 의하여 정의된 렬에서 p가 참일 때 첫 출현에 대한 반복자를 돌려 준다. 렬에서 출현이 없다면 e를 돌려 준다.

template <**class** Iter,**class** pred>

Iter lower_bound(Iter s ,Iter e,**const** T &V)

앞의 반복자 s와 e에 의하여 정의된 분류에서 v값의 첫 출현에 대하여 지적하는 반복자를 돌려 준다. 만일 렬에서 v가 나타나지 않으면 v보다 큰 첫 요소에 대하여 지적하고 반복자를 돌려 준다(이러한 요소가 없다면 표에 대한 지적자를 돌려 준다).

template <**class** T>

const T& max (**const** T &a ,**const** T &b)

a와 b중에서 최대값을 돌려 준다.

template <**class** In1,**class** In2,**class** out>

out merge(In1 s1,In1,e1,In2 s2,In2 e2,out s3)

입력반복자 s1과 e1에 의하여 정의된 분류값의 렬은 분류된 값의 결합순차를 제시하도록 s2

와 e2에 의하여 정의된 분류값의 렬과 통합된다. 결합된 순차는 출력반복자 s2가 지시하는 위치에서 분류된다. 함수의 귀환값은 결합된 순차의 표기에 대한 반복자이다.

template <**class** T>

const T& min(**const** T &a,**const** T &b)

a와 b중에서 최소값을 돌려 준다.

template <**class** Iter>

void random_shuffle(Iter s,Iter e)

임의접근반복자 s와 e에 의하여 정의된 요소들의 순차모조란수방식에서 배렬한다.

template <**class**Iter, **class** T>

Iter remove(Iter s, Iter e, **const** T &y)

앞의 반복자 s와 e에 의해 정의된 순차에서 나타난 v값을 삭제한다. 함수는 압축된 순차의 표기에 대하여 지정하는 반복자를 돌려 준다.

template <**class** Iter,**class** T>

void replace (Iter s,Iter e,**const** T &V,**const** T &w)

앞의 반복자 s와 e에 의하여 정의된 순차에서 나타난 v값을 w값으로 교체한다.

template <**class** Iter>

void reverse (Iter s ,Iter e)

쌍방향반복자 s와 e에 의하여 정의된 요소들의 순차를 교환한다.

template <**class** T>

void swap(T &a,T &b)

객체 a와 b의 값을 서로 바꾼다.

template <**class** Iter,**class** T>

Iter unique(Iter s, Iter e)

앞의 반복자 s와 e에 의하여 정의된 순차에서 값이 같은 요소들의 묶음의 첫 요소를 제외하고 모두 삭제한다. 함수의 귀환값은 압축된 순차의 표기에 대한 반복자이다. 함수의 귀환값은 수렬의 시작에 대한 순환자이다.

template <**class** Iter **class** T>

Iter upper_bound(Iter s ,Iter e,**const** T &V)

앞선 반복자 s와 e에 의하여 정의된 분류순차에서 값 v의 마지막출현에 대하여 지적하고 반복자를 돌려 준다. 순차에서 v가 나타나지 않으면 v보다 큰 첫 요소에 대하여 지적하는 반복자로 돌려 준다(이러한 요소가 없다면 표기에 대한 지적자를 돌려 준다).

부록 3. 표준클라스들

C++표준은 100개이상의 표준클라스와 구조체들을 정의한다. 이 부록에서는 표준본보기서고용기와 문자렬클라스에 대하여서만 볼수 있다. 표준클라스들에 관한 구체적인 정보에 대하여서는 다음의 참고서를 보시오.

- B.Stroustrup, The C++ Programming Language, 3rd ed.,Reading,MA: Addison-Wesley, 1998.
- X3 Secretariat, Draft Standard-The C++ Language, X3J16/97-14882, Washington, DC: Information Technology Council (NSITC), 1997.
- P.J.Pluger(editor), A.A.Stepanov, M.Lee, and D.R.Musser, The C++ Standard Template Library, Englewood Cliffs, NJ:Prentice-Hall, 2000.

3.1 용기클라스

표준본보기서고의 용기클라스는 프로그람작성자가 개별적목록에서 얻은 요소의 형을 렬거하는 일반목록표현들의 집합이다. 묶음을 제한한 공간에로 용기클라스들은 확장될수 있다. 실례로 자동적으로 첨자를 검출하는 정의된 용기클라스를 만들수 있다. 8개의 기본용기클라스들이 있다. 그중에서 5개는 목록을 요소들의 렬로 취급한다. 이것을 제공하는 용기들은 deque, list, priority_queue, queue, stack, vector들이다. 이 2개의 용기들은 더 좋은 편관방법으로 목록을 본다. 다른 두개의 용기클라스들은 map와 set이다. 이러한 2개의 용기들은 많은 편관방식에서 목록을 본다. 8개 클라스에 대한 설명은 다음에 한다. 클라스 priority_queue, queue, stack는 다른 용기를 사용하여 이 클라스를 만들기때문에 용기적응기 혹은 적응기라고 한다.

deque

렬에서 개별적요소들에 대한 그 순간의 임의접근을 제공한다. 추가적으로 deque는 그 시간에 렬의 시작 혹은 마지막으로부터 삽입하거나 지울수 있다.

list

렬에서 개별적요소들에 대한 그 순간의 순차접근을 제공한다. 추가적으로 list는 그 시간에 렬안의 임의의 위치에서 요소를 삽입하거나 지울수 있다.

priority_queue

우선권에 기초한 접근을 제공한다. priority_queue는 가장 높은 우선권을 가진 요소에 대한 정돈된 로그시간적접근을 제공한다. 추가적으로 priority_queue는 정돈된 로그시간에 렬안의 임의의 위치로부터 요소를 삽입하거나 지울수 있다.

queue

선입선출방식의 요소접근을 제공한다. queue는 렬의 시작이나 끝에 대한 그 순간의 접근을 제공한다. 추가적으로 queue는 렬의 마지막에 삽입할수도 있고 그 순간에 렬의 시작부터 지울수 있다.

stack

후입선출방식의 요소접근을 제공한다. stack는 렬의 마지막에 대한 그 순간의 접근을 제공한다. 추가적으로 stack는 렬의 마지막에 요소를 넣거나 지울수 있다.

vector

렬에서 개별적인 요소들에 대한 그 순간의 임의접근을 제공한다. 정돈된 그 시간에 vector는 렬의 마지막에 삽입하거나 지울수 있다. 다른곳에 대한 삽입 혹은 삭제에는 렬의 크기에 비례되는 시간이 필요하다.

map

목록에서 개별적요소들에 대한 그 순간의 순차접근을 제공한다. 유일한 열쇠값은 매 요소값에 관계된다. 열쇠값에 기초한 요소에 대한 접근은 로그시간동안에 진행된다.

set

목록에서 개별적요소들에 대한 그 순간의 순차접근을 제공한다. 그 값에 기초한 요소에 대한 접근은 로그시간에 진행될수 있다.

련관 없는 용기클라스본보기들은 일반적으로 한두개의 파라메터를 취한다. 첫 본보기파라메터는 항상 용기가 가지고 있는 값의 형이다. 두번째 파라메터가 제공되면 그것은 목록의 요소를 위한 기억기할당방법을 수행하는 클라스이다. 기정기억기할당방법은 대부분의 프로그람작성위치에서 진행되므로 표현되는 임의의 할당파라메터를 무시한다.

련관 있는 용기클라스본보기들은 한개 또는 두개, 세개의 본보기파라메터(두번째, 세번째 파라메터는 선택적이다.)를 취할수 있다. 첫번째 파라메터는 용기가 가지고 있는 값의 형이다. 두번째 파라메터는 목록요소비교를 진행하는 클라스이다. 세번째 파라메터는 기억기할당파라메터이다. 용기에 대한 론의에서는 초보적인 vector클라스에 대하여 취급한다. 그것이 가장 유력한 목록표시이기때문에 다른 클라스에 대한 론의에서는 vector에 대한 차이점과 류사성을 강조한다.

3.1.1 용기벡토르

본보기추상자료형 vector<T>는 배렬표시법을 리용한 요소의 목록을 표시할수 있는 객체를 지원한다. 이 클라스의 객체와 다음부분의 string클라스는 결국 습관적인 변환배렬과 문자렬들의 대부분의 사용을 모두 바꾼다. 클라스본보기 vector의 표시는 기억기할당에 대하여 임의의 본보기파라메터를 무시하는데서 간단하게 한다. 다음의 목록은 선택된 vector<T>성원함수와 연산자를 서술한다.

설명에서 size_type는 부호 없는 옹근수형이며 iterator는 임의접근반복자이며 reference는 T&로 전환시킬수 있는 형이며 **const**_reference는 **const** T&로 전환될수 있는 형이다. 이러한 형들은 vector<T>를 위한 클라스정의에서 선언된다.

vector::vector()

기정구축자는 0길이의 벡토르를 창조한다.

vector::vector(**const** T &V)

복사구축자는 벡토르 v와 같은 벡토르를 창조한다.

vector::vector(size_type n, **const** T &Val=T())

명백한 구축자는 매 요소가 val로 초기화된 n길이의 벡토르를 창조한다.

vector::~vector()

해체자는 벡토르를 위한 동적기억기를 해제한다.

reference vector::at(**int** i)

i가 유효한 첨수라면 i번째 요소를 돌려 주며 아니면 례외가 발생한다. 귀환요소는 수정될 수 없다.

reference vector ::back ()

벡토르의 마지막요소에 대한 참조를 돌려 준다.

const_reference vector::back() **const**

벡토르의 마지막요소에 대한 상수참조를 돌려 준다.

iterator vector::begin()

벡토르의 첫 요소를 지적하는 반복자를 돌려 준다.

const iterator vector::begin()

vector의 첫 요소를 저적하는 반복자를 돌려 준다. 반복자에 의하여 참조되지 않은 요소들은 수정할수 없다.

void vector::clear()

벡토르로부터 모든 요소를 제거한다.

bool vector::empty() **const**

벡토르에 요소가 없으면 참을 돌려 주고 아니면 거짓을 돌려 준다.

iterator vector::end()

벡토르의 마지막요소다음위치를 직접 지적하는 반복자를 돌려 준다.

const _iterator vector::end()

마지막요소다음위치를 직접 지적하는 반복자를 돌려 준다. 이 반복자에 의하여 참조되지 않은 요소들은 수정할수 없다.

iterator vector::eraser(iterator pos)

pos위치에서 벡토르의 요소를 제거한다. 벡토르에서 복사의 위치를 돌려 준다.

reference vector::front()

벡토르의 첫 요소에 대한 참조를 돌려 준다.

const _reference vector::front() **const**

벡토르의 첫 요소에 대한 상수참조를 돌려 준다.

iterator vector::insert(iterator pos,**const** T &Val=T())

벡토르의 pos위치에 val을 복사한다. 벡토르에서 복사위치를 돌려 준다.

vector<T>& vector::**operator** =(**const** vector <T> &V)

성원할당연산자는 벡토르 v의 복사벡토르를 만든다. 수정된 벡토르를 돌려 준다.

reference vector::**operator**[](size_type I)

벡토르의 i번째 요소에 대한 참조를 돌려 준다.

const_reference vector::**operator**{} (size_type i)**const**

벡토르의 i번째 요소에 대한 상수참조를 돌려 준다.

void vector::pop_back()

객체의 마지막요소를 제거한다.

void vector::push_back(**const** T &Val)

객체의 마지막요소다음에 val을 복사한다.

reverse_iterator vector ::rbegin()

벡토르의 마지막요소를 지적하는 반전반복자를 돌려 준다.

const_reverse_iterator vector::rbegin()

벡토르의 마지막요소를 지적하는 반전반복자를 돌려 준다. 이 반복자에 의하여 참조되지 않은 요소는 수정할수 없다.

iterator vector::rend()

첫 요소보다 앞선위치를 직접 지적하는 반전반복자를 돌려 준다.

const_reverse_iterator vector ::rend()

첫 요소의 앞위치를 직접 지적하는 반전반복자를 돌려 준다. 이 반복자에 의하여 참조되지 않은 요소는 수정할수 없다.

void vector::resize(size_type s,T val=T())

n은 벡토르에서 요소의 수이다. s>n이면 요소의 수는 va1을 가진 새 요소들의 초기값과 존재하는 요소다음에 추가된 새 요소와 함께 총수가 s에 이르도록 증가된다. s<n이면 요소의 수는 벡토르의 마지막으로부터 요소를 지워 총수가 s에 이르도록 감소된다. s와 n이 같으면 아무 동작도 하지 않는다.

size_type vector::size() **const**

벡토르에서 요소의 수를 돌려 준다.

void vector:: swap(vector<T> &V)

현재의 벡토르와 벡토르 v를 바꾼다. 이 연산은 요소들의 개별적인 바꾸기보다 훨씬 더 효과적이다. 벡토르서고는 또한 동등성과 관계연산자를 다중정의한다.

bool operator==(**const** vector <T> &U,**const** vector <T> &V)

벡토르 U와 V가 같은 크기를 가지며 호상관계를 가지는 요소들이 갈다면 참을 주고 아니면 거짓을 돌려 준다.

bool operator!=(**const** vector<T> &U,**const** vector <T> &V)

!(U==V)를 돌려 준다.

bool operator <(**const** vector<T> &U,**const** vector <T> &V)

연산자는 V에서 요소의 렬이 V에서 요소의 적당한 초기첨수이라면 참을 돌려 준다(즉 U[i]는 i에 대하여 0…U.size()-1사이와 U.size()<v.size()상태에서 갈다).

연산자는 또한 U[i]가 V[i]보다 작고 U[I], U[2], …, U[i-1]과 V[i], V[2], …, V[i-1]이 같은 0…U.size()-1사이에서 i가 존재한다면 참을 돌려 준다. 그렇지 않으면 거짓을

돌려 준다.

bool operator <=(const vector <T> &U, const vector <T> &V)

!(V<U)를 돌려 준다.

bool operator >(const vector <T> &U, const vector <T> &V)

(V<U)를 돌려 준다.

bool operator >=(const vector <T> &U ,const vector <T> &V)

!(U<V)를 돌려 준다.

3.1.2 list용기

list용기는 첨자지정과 at()성원함수를 제외한 벡토르용기를 위하여 모든 기능을 제공한다. 이 동작은 list반복자가 임의접근을 지원할수 없기때문에 제공되지 못한다. 0list용기는 목록을 결합하는 성원함수 splice()와 merge()와 목록을 재배치하는 성원함수 sort()를 제공한다. 추가적으로 성원함수 push_front()와 pop_front()는 목록의 첫 요소를 추가하고 제거한다.

3.1.3 deque용기

deque용기는 vector와 list클라스의 혼합과 같다. 그것은 vector와 같은 동작을 지원하며 목록의 첫 요소들을 추가, 삭제하기 위한 list동작 push_front()와 pop_front()를 지원한다.

3.1.4 map용기

map용기는 열쇠에 기초한 한쌍(열쇠, 값)의 효률적인 복귀를 형식으로 제공한다(열쇠들은 하나만 필요하다). 효률적인 복귀는 map의 요소들이 열쇠에 기초한 배렬순서에서 지속되기때문에 가능하다. Map용기는 표준반복자성원함수 begin(), end(), rbegin(), rend()를 제공한다. 첨자지정연산자는 열쇠에 기초한 복귀와 삽입을 수행하도록 다중정의된다. 성원함수 find(), count(), lower_bound(), upper_bound()는 열쇠에 기초한 요소의 찾기와 계수를 한다. 추가적으로 전통적인 용기성원함수 insert(), erase(), empty(), size(), swap()가 있다.

3.1.5 set용기

set용기는 열쇠를 제외하고 map와 같다. 열쇠가 없으므로 성원첨자지정연산자는 없다. 이 성원연산자가 아니라 map와 set의 대면부를 비교할수 있다.

3.1.6 queue적응기용기

queue는 선입선출방식(FIFO:first-in-first-out)을 제공하는 적응기클라스이다. 구체적으로 특히 처음과 마지막요소에 접근하기 위한 성원함수 front()와 back(), 새로운 마지막요소를 삽입하기 위한 push(),첫 요소를 제거하기 위한 pop()들이 있다. 추가적으로 전통적인 용기성원함수 empty()와 size()가 있다. queue를 위한 기정실행은 deque를 사용한다.

3.1.7 priority_queue적응기용기

priority_queue는 우선권에 기초한 요소의 제거를 제공하는 적응기클라스인데 요소의 우선권은 그

값에 비례한다. 구체적으로 우선권이 가장 높은 요소에 접근하기 위한 성원함수 top(), 새 요소를 삽입하기 위한 push(), 가장 높은 우선권을 가진 요소를 제거하기 위한 pop()들이 있다. 추가적으로 전통적인 용기성원함수들로서 empty()와 size()가 있다. priority_queue의 기정실행은 vector를 사용한다.

3.1.8 stack적응기용기

stack는 후입선출방식(LIFO;Last-in-first-out)의 요소접근을 제공하는 적응기클라스이다. 구체적으로 마지막요소에 접근하기 위한 성원함수 top(), 새로운 마지막요소를 삽입하기 위한 push(), 마지막요소를 제거하기 위한 pop()들이 있다. 추가적으로 전통적인 용기성원함수 empty()와 size()가 있다. stack()의 기정실행은 deque를 사용한다.

3.2 string클라스

표준적으로 클라스 string은 본보기클라스 basic_string의 실체화된 판본이다.

typedef basic_string <char> string;

이 추상자료형은 문자들의 렬을 표시할수 있는 객체를 지원한다(렬은 임의의 길이를 가진다). 프로그람실행시 string객체는 여러가지 서로 다른 길이의 렬을 표시할수 있다.

3.2.1 문자렬성원함수

다음목록은 선택된 string성원함수와 연산자를 서술한다. 설명에서 size_type는 부호 없는 옹근수형이며 iterator는 임의접근반복자이며, reference는 T&로 변환할수 있는 형이며 const_reference는 const T&로 변환할수 있는 형이다. 이 형들은 string을 위한 클라스정의에서 선언된다. 일부 성원함수들은 클라스에서 정의된 옹근수형상수 npos를 사용한다. 상수는 개별적으로 0…n-1사이의 밖에 있는 값을 가지는데 n은 표시하는 문자렬에서 문자수이다.

string::string()

기정구축자는 빈 문자렬을 표현하기 위하여 초기화한다.

string::string(const char s[])

빈 완료문자 char형 s배렬에 복사를 표시하기 위하여 객체를 초기화한다.

string& string ::append(const char s[])

현재문자렬의 끝에 빈 완료배렬 s의 복사를 추가한다. 수정된 현재문자렬이 돌려 진다.

string& string::append(const string &s)

현재문자렬의 끝에서 문자렬 s를 복사해 넣는다. 수정된 현재문자렬이 돌려 진다.

const_reference string::back() const

문자렬의 마지막요소에 상수값을 준다.

reference string:: back()

문자렬의 마지막요소에 값을 준다.

iterator string::begin()

문자렬의 첫 요소를 가리키는 반복자를 준다.

const char *string::c-str() const

요소들이 표현되는 문자렬의 복사인 char형배렬의 개별요소들을 준다. NULL문자는 복사를 끝낸다.

int string ::compare(const string &s,size_type n) const

문자렬 s로 문자렬의 시작부터 비교한다. 비교부분이 같으면 0을 주고 비교부분에서 먼저 수정하여야 한다면 부수값을 주고 아닌 경우 정수값을 준다.

const char *string::data() const

요소가 표시문자렬의 복사인 char형배렬의 개별요소들을 준다.

bool string::empty () const

문자렬값이 빈 문자렬이면 참을 주고 아니면 거짓을 준다.

iterator string :: end()

문자렬의 마지막요소다음을 가리키는 반복자를 준다.

string& string :: erase(size_type n,size_type m)

n부터 m까지의 문자를 문자렬로부터 없앤다. 문자렬의 수정값을 준다.

size type string :: find(const string &s, size-type n=0)

const string &s와 size-type n=0은 보조문자렬을 위해 n위치를 시작탐색으로 하여 현재문자렬에서 오른쪽을 탐색한다. 보조문자렬을 발견한다면 그 함수는 보조문자렬에 앞서 첫번째 발견의 시작위치를 돌린다. 보조문자렬이 발견되지 않았다면 상수 npos를 돌린다.

const_reference string :: front() const

문자렬의 첫번째 요소에 대한 상수참조를 돌려 준다.

reference string :: insert(size_type n const string &s)

현재문자렬의 n과 n+1의 위치들사이에 s문자렬의 복사를 돌린다. 수정된 현재문자렬이 돌려 진다.

const_reference string :: operator[](size_type i) const

i가 size()보다 작다면 그 함수는 문자렬로 표시된 i번째 문자로 복사를 돌린다. i 가 size()와 같다면 0을 돌린다. 그렇지 않으면 동작이 정의되지 않는다.

reference string :: operator[](size_type)

i가 size()보다 작다면 그 함수는 표시된 i번째 문자렬로 참조를 돌린다. 그렇지 않으면 동작이 정의되지 않는다.

size_type string :: rfind(const string &s, size_type n=npos) const

n위치에서 시작한 탐색으로서 보조문자렬 s를 위하여 현재문자렬에서 왼쪽으로 탐색한다. 보조문자렬이 발견되면 그 함수는 보조문자렬의 첫번째로 발견한 시작위치를 돌린다. 보조문자렬이 발견되지 않았다면 정수 npos를 돌린다.

size-type string :: size() **const**

문자렬의 길이를 돌린다.

3.2.2 string보조함수

서고는 또한 삽입, 추출, 추가, 관계연산자들을 다중정의한다.

iostream& **operator** >> (istream & sin, string &s)

입력흐름 sin으로부터 다음번 비공백문자를 추출하고 그것을 s에 할당한다. sin에 대한 참조를 돌려 준다.

stream& **operator** << (ostream & sout, string &s)

sout출력흐름에 s문자렬을 삽입한다. sout에 대한 참조를 돌려 준다.

string **operator** + (**const** string &s, **const** string &t)

문자렬 s와 t의 련속인 문자렬을 돌린다.

string **operator** + (**const** string &s, **const char** t[])

문자렬 s와 무효로 끝난 문자렬 t의 련속인 문자렬을 돌린다.

string **operator** + (**const chars**[], **const** string &t)

무효로 끝난 문자렬 s와 t문자렬의 련속인 문자렬을 돌린다.

bool operator = = (**const** string &s, **const** string &t)

s.compare(t)== 0을 돌려 준다.

bool operator !=(**const** string &s,**const** string &t)

s.compare(t)==0을 돌려 준다.

bool operator <(**const** string &s,**const** string &t)

s.compare(t)<0을 돌려 준다.

bool operator<=(**const** string &s,**const** string &t)

s.compare(t)<=0을 돌려 준다.

bool operator>(**const** string &s,**const** string &t)

s.compare(t)>0을 돌려 준다.

bool operator>=(**const** string &s,**const** string &t)

s.compare(t)>=0을 돌려 준다.

부록 4. 개선된 기능들

이 부록에서는 3개의 개선된 기능들인 이름공간,례외, 동료(friend)들에 대하여 고찰한다. 이름공간은 클라스들과 함수들, 객체들, 형들의 이름 붙은 모임이다. 이름공간은 여러가지 다중정의로 된 클라스들과 함수들을 식별하는 의뢰기프로그람들을 위하여 필요하다.

례외는 실행시에 발생하는 프로그람오유이다. 례외가 발생하고 례외처리기의 코드토막이 례외를 나타내는 상태로 되였다면 조종흐름은 조종기로 전송된다. 흔히 쏘프트웨어는 불만족한 동적할당요구, 연산오유(실례로 0으로 나누기), 예상치 않았던 입력과 적당치 않은 배렬첨수지정과 같은 사건들에 대한 례외발견과 처리기능을 제공한다. 끝으로 C++ 의 friend속성을 고찰한다. friend속성은 선택된 비성원함수나 클라스에 의하여 자료성원에 접근할수 있는 제한된 조건에서만 리용할수 있다.

4.1 이름공간

C++는 여러가지 각이한 령역, 더 정확하게는 이름공간들을 인식한다. 실례로 요소(클라스, 함수, 객체)들이 선언되는 곳에서 블로크로 제한되는 국부이름공간이 있다. 다른 기본이름공간은 전역이름공간과 std이름공간이다. std이름공간은 표준서고클라스들과 함수들이 존재하는 이름공간이므로 실례를 통하여 이것을 리용하였다. 다중정의규칙에 의하여 주어 진 이름공간과 제목에 있는 요소들의 이름은 일치하다.

표준C++는 다른 이름공간들을 정의 및 참조하는 namespace와 using기구를 포함한다. 하나의 이름공간안에서 클라스, 함수, 객체형들의 집합체와 다른 이름공간들을 선언할수 있다.

4.1.1 정의

다수의 서고들은 응용프로그람개발에서 자주 요구된다. 이 서고들은 다른 원천에서 넘어 올수 있으므로 같은 이름으로서 클라스들이나 함수들을 정의하여야 한다. 다음의 머리부파일 lib1.h에 서고 lib1가 있다고 가정한다.

```
#ifndef LIB1_H
#define LIB1_H
class SimpleWindow {
    // EzWindows창문표현
};
#endif
```

같은 종류로 다음의 머리부파일 1:b2.h에 서고 1:b 2가 있다고 가정한다.

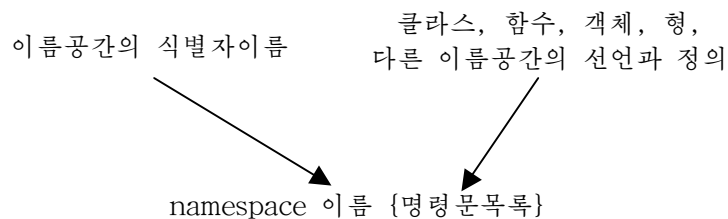
```
#ifndef LIB2_H
#define LIB2_H
class SimpleWindow {
    //집창문표현
};
#endif
```

lib1과 lib2는 SimpleWindow클라스를 정의한다. 의뢰기응용프로그람이 다음의 코드토막에서처럼 같은 프로그람파일에서 두개의 클라스들을 다 리용하려고 한다면 어떤 현상이 나타나는가?

```
#include “lib1.h”
#include “lib2.h”
void BuildHouse() {
    SimpleWindow  FigureWindow; //lib1 혹은 lib2인가?
    SimpleWindow StormWindow;   //lib1 혹은 lib2인가?
    //객체를 처리한다.
}
```

함수 BuildHouse()는 FigureWindow 와 StormWindow 정의에 리용된것이 어느 SimpleWindow 클라스인가가 명확치 않기때문에 콤파일될수 없다. 만일 lib1과 lib2서고를 이름공간기술에 의하여 개발되였다면 BuildHouse()를 재정의하는 다음의 코드토막에서와 같이 SimpleWindow클라스를 개별적으로 지정하여야 한다.

```
#include “lib1.h”
#include “lib2.h”
void BuildHouse() {
    lib2::SimpleWindow FigureWindow; //lib1
    lib1::SimpleWindow StormWindow;  //lib2
    //객체를 처리한다.

}
```

이름공간들은 요소들의 명확치 않은 이름을 방지하도록 렬거할수 있는 가상프로그람들을 창조한다. 이름공간을 정의하는 기본문법은 아주 간단하다.

SimpleWindow클라스만 리용하여 Build&Use()함수의 판본을 수정할수 있도록 이름공간 l:b1를 리용하는 서고 l:b1에 대한 머리부파일을 재정의한다.

```
#ifndef LIB 1_H
#define LIB 1_H
namespace lib 1 {
    class SimpleWindow {
        //EzWindows창문표현
    };
}
# endif
```

lib2이름공간을 리용하는것처럼 서고 lin2의 머리부파일을 비슷하게 재정의한다.

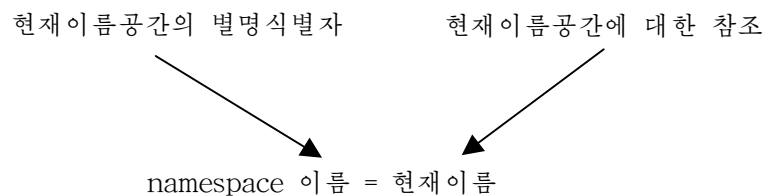
```
#ifndef LIB 2_H
#define LIB 2_H
namespace lib 2 {
    class SimpleWindow {
        //집창문표현
    };
}
#endif
```

이름공간의 이름은 자기의 머리부파일이름에서 대표적인 변화이다. 이름공간의 이름의 특수성을 담보하기 위하여 때때로 쏘프트웨어제작자들은 이름공간의 이름에 부분적으로 서고의 이름을 포함시킨다.

```
namespace FantasticFunctionsClassyClassesLib {
    //이름공간정의
}
```

의뢰기는 이름공간의 별명을 리용하여 필요 없는 이름의 리용을 피할수 있다.

```
namespace lib = FantasticFunctionsClassyClassesLib;
```

실례에서 보여 준것처럼 이름공간별명은 다음과 같은 문법을 가진다.

이름공간은 이름을 가질것을 요구하지 않는다. 이러한 이름공간이 닉명이름공간이다. 이름을 밝히지 않은 이름공간은 리용할 때 대체로 번역단위(프로그람파일)의 시작에서 정의된다. 이름을 밝히지 않은 이름공간의 요소들은 같은 번역단위에서만 참조될수 있다. 이름을 밝히지 않은 이름공간의 요소는 이름공간을 생기게 하는 번역단위를 위한 전역이름공간이다. 따라서 이름을 밝히지 않은 이름공간의 요소들은 자기의 이름만을 리용하여 참조된다. 다음의 실례에서 객체 MaxSize는 이름을 밝히지 않은 공간에서 정의되였으며 배렬 s를 정의하는 함수 f()안에서 리용된다.

```
namespace {
    const int MaxSize = 256;
}
void f() {
    int s[MaxSize];
    //s를 처리
}
```

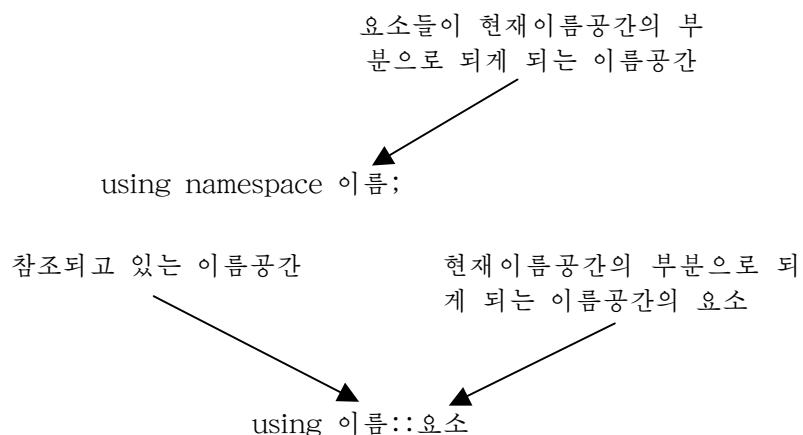
이름공간에는 추가적인 경우와 보충적인 선언을 가질수 있다. 또한 이름공간정의는 겹싸여 질수 있

다. 실례로 다음의 코드토막은 처음에 함수 f()를 포함하는 이름공간 A를 정의한다. 함수 g()는 그다음에 정의된다. 이름공간 B는 그다음에 함수 h와 함수 i()를 포함하는 이름공간 c를 포함한다. 이름공간 A는 그다음 함수 j()를 포함하기 위하여 수정된다.

```
namespace A {
void f ( ) ;
}
void g ( ) {
    //이름공간 A의 함수 f()를 사용할수 있다.
    //이름공간 B는 전혀 사용할수 없다.
    // 이름공간 A의 함수 g()를 사용할수 없다.
}
namespace  B {
    void h();
    namespace C {
        void i();
    }
}
namespace  A {
    void j ( );
}
```

우의 실례에서 함수 g()는 번역될 때 선언되지 않았기때문에 이름공간 B의 부분품을 리용할수 없다. 이러한 리유로 하여 g()함수는 이름공간 A의 함수 j()를 리용할수 없다.

4.1.2 이름공간의 리용

using명령문은 자주 이름공간으로 리용된다. 이 명령문은 실제로 어느것이 선언되는가를 지적하기 위한 기구이다. using명령문에는 두가지 형태가 있다.

실례로 다음과 같은 정의가 있다.

```
namespace A {
```

```
        f() {
            cout << "f(): from namespace A" << end1;
        }
        void g() {
            cout << "g(): from namespace A" <<end1;
        }
    namespace B {
        void f() {
            cout << "f(): from namespace B" << end1;
        }
        namespace C {
            void g() {
                cout << "f(): from namespace C" <<end1;
            }
        }
    }
    void g() {
        cout << "g(): from global namespace" << end1;
    }
```

다음의 코드토막이 실행되면 어떤 현상이 생기는가?

```
g();
A::f();
B::f();
B::C::f();
using namespace A;
f();
```

출력결과는 다음과 같다.

```
g(): from global namespace
f(): from namespace A
f(): from namespace B
f(): from namespace C
f(): from namespace A
```

이름공간이 아닌 정의들이 선행되였기때문에 실례에서 함수 g()에 관해서는 애매한것이 없다. 실례에서 호출 A :: f()는 유효범위해결연산자의 사용이 충분히 제한되였기때문에 명백하다. 호출 B::f<>은 역시 이름공간 C내부안에서 선언된 다른 함수 f()가 참조된 추가적인 제한을 요구하기때문에 명백하다. 호출 B::C::f()는 유효범위해결연산자의 사용이 충분히 제한되였기때문에 대체로 명백하다.

실례에서 using명령문은 이름공간 A에서의 정의가 현재이름공간의 부분이라는것을 가리킨다.

```
using namespace A ;
```

이름공간에 존재하는 함수들은 애매한것이 없기때문에 using명령문다음의 함수 f()를 호출하는데는 그 어떤 제한이 없다. 그러나 명령문아래부분은 콤파일되지 않는다.

```
g( ) ;
```

현재이름공간에서 같은 특징을 가진 두개의 함수 g()가 있기때문에 애매한것이 있을수 있다. 두 함수를 식별할수 있도록 유효범위해결연산자를 사용할수 있다.

```
::g( ) ;   //전역이름공간의 g( )
A::g( ) ; //이름공간 A의 g( )
```

실례로 다음의 코드토막에서와 같이 이름공간 A보다 오히려 이름공간 B의 요소이름공간 C를 리용한다면

```
g();
A::f();
B::f();
B::C::f();
using namespace B::C;
f();
```

이다. 이 토막에서 결정적호출은 출력에서 나타난다.

```
f() : form namespace C
```

출력은 지금 C에서의 정의가 현재이름공간에서 다른 이름을 가지고 선행한것과 같으므로 나타난다. 코드토막에서는 여러가지 객체정의를 포함한 이름공간 constants를 정의한다.

```
namespace constants {
    const double pi = 3.141592 ;
    const double e = 2.718281 ;
    const double c = 299792.4 ;

}
```

다음의 토막에서 using명령문은 이름공간 constants에서 pi만을 접근하도록 한다.

```
using constants :: pi ;
double r ;
cout << “circle radius :” << flush ;
cin >>r ;
double c = 2 * pi * r ;
cout << ” circumference of circle with radius”
  << r  << “ : ” <<C <<end1;
```

그 결과는 객체 C의 정의가 문장론적인 애매성을 나타내지 않는다는것을 보여 준다.

4.2 례외처리

례외는 실행시에 발생하는 프로그람오유이다. 례외가 발생하고 례외처리코드토막이 실제로 례외처리를 위한것이라면 조종흐름은 처리기로 돌아 온다. 그대신 례외가 발생하고 그를 위한 처리기가 없다면 프로그람은 완료된다.

4.2.1 기초

8장에서 Rational ADT를 개발하였다. 특히 옹근수파라메터 denom이 분모의 새로운 값이라고 생각하는 변이자성원함수를 개발하였다. 아래에 이에 대한 함수정의가 있다.

```
void Rational :: SetDenominator (int denom) {
    if (denom != 0) {
        DenominatorValue = denom;
    }
    else {
        cerr << "Illegal denaminator" : <<denom <<" using1 "<< end1;
        DenominatorValue = 1;
    }
}
```

함수 SetRational()은 denom값을 검사한다. denom이 0이 아니라면 자료성원 DenominatorValue는 denom으로 설정된다. 그밖의 오유통보가 표시되면 DenominatorValue값은 1로 설정된다. SetDenominator()함수의 선택실행을 목록 4-1에 주었다. 이 판본이 분모를 0으로 설정하려고 시도한다면 례외가 발생한다.

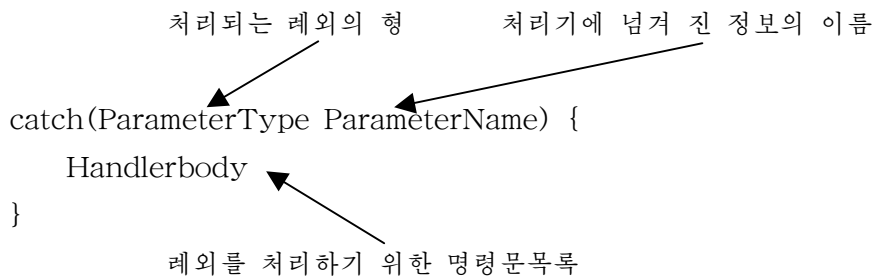
목록 4-1. 잘못된 분모변경의 처리와 발견

```
void Rational  :: SetDenominator(int denom) {
    try {
        if(denom != 0) {
            DenominatorValue = denom;
        }
        else {
            throw(denom);
        }
    }
    catch(int d) {
        cerr << “Illegal denominator:” <<denom << □using 1□ << end1;
        DenominatorValue = 1;
    }
}
```

례외처리에는 3가지 구성요소가 있다. 하나는 시도(trying)이고 다른것들은 내보내기(throwing), 포착(catching)이다.

코드는 try블로크에서 함수를 인용하여 간접 혹은 비간접적으로 례외가 발생되는 상태를 처리한다. try블로크는 trying의 예약어에 있는 명명문블로크이다. try블로크안의 명령문들에서 하나는 throw명령문으로 될수 있다.

throw명령문은 함수호출과 류사하다. 실례로 틀린 상태가 발견되면 throw명령문은 괄호안에 나타난 오유정보를 호출한다. 정보의 복사는 throw명령문을 통하여 례외처리기에 전달된다.

다음의 try블로크는 catch례외처리기이다. 이것은 일반적으로 try블로크에서 발생할수 있는 여러가지 례외처리를 위한 catch처리기이다. catch처리기지정은 함수정의와 류사하다.

catch처리기는 열쇠단어 catch와 단일파라메터목록으로 시작된다. 파라메터형은 try블로크에서 발생될수 있는 례외의 형과 조화된다. 다음의 파라메터목록은 명령문블로크이다. catch처리기가 례외를 처리하기 위하여 호출되면 명령문블로크가 실행된다.

목록 4-1의 SetDenominator()함수에서 try블로크는 denom값을 검사한다.

```
try {
    if (denom != 0) {
        DenominatorValue = denom;
    }
    else {
        throw(denom);    }
}
```

만일 denom값이 0이면 례외가 발생한다.

```
throw (denom);
```

이때 례외값은 0(denom)으로 된다.

int를 처리하는 catch처리기를 정의하였기때문에 잘못된 분모에 대한 례외처리기이다.

```
catch ( int d) {
    cerr << "Illegal denominator:" <<d
      << "using 1" <<end1;
    DenomivatorValue = 1;
}
```

분모처리기는 오유통보에서 례외값을 현시하고 분모값을 1로 설정한다. 목록 4-1의 SetDenominator()함수는 8장에서와 같다.

례외처리수법은 객체지향프로그람들에서 필요한 수법이다. 례를 들어 함수 h()에 의하여 함수 g()가 호출되고 그에 의하여 호출된 f()에서 오유가 발생하였다고 하자. 오유가 정확하다면 이 호출순서를 일으켰던 함수 h()는 조종흐름을 회복해야 한다. 례외처리수법은 이러한 유연성을 가지고 있다. 즉 그것은 함수호출을 중지할수 있고 오유로 되는 문제들이 정확한 동작을 하게 한다. 수정이 끝나면 조종명령은 례외를 처리하는 catch처리기목록다음의 명령문을 수행한다. 이에 대한 실례를 더 들어 보자.

8장의 Rational의 실행에서 SetDenominator()함수는 Rational구축자들과 변환기 Extract()에 의하여서만 호출된다. 목록 4-1에서와 같이 SetDenominator()대신에 특정한 함수처리를 진행하며 례외를 처리하는 성원함수들을 리용할수 있다. 이 과정에 대한 실례를 목록 4-2에서 준다. 목록 4-2에서 SetDenominator()함수의 수정된 정의는 분모값이 0이라면 례외가 발생한것으로 되지만 례외를 처리하지는 않는다.

례외가 발생하면 조종흐름은 즉시에 함수 SetDenominator()에서 호출함수로 넘어 간다. 만일 호출함수가 례외를 처리하지 않으면 조종흐름은 곧 그 함수에로 넘어 간다. 재귀적처리는 계속 호출되는 함수들가운데서 어느 하나가 례외를 처리할 때까지 함수호출이 계속된다. 이것이 후에 발생하면 프로그람은 완료된다. 실례로 목록 4-2의 Rational구축자 혹은 Extract()성원함수가 호출되면 례외가 처리된다. 구축자가 례외를 처리하면 오유통보가 현시되며 SetDenominator()함수는 합법적인 분모값으로 다시 호출된다.

```
catch (int d) {
    cerr << “Illegal denominator:” << d
      << “using 1” << end1;
    SetDenomivator(1);
}
```

Extract()성원함수가 례외를 처리하면 오유통보가 현시되며 Extract()는 합리적인 값을 골라 내기위하여 재귀호출된다.

```
catch (int d) {
    cerr << “Illegal denominator:” << d << end1;
    cerr << “Reenter rational value:” ;
    Extract(sin);
}
```

목록 4-2. 잘못된 분모를 위한 특수한 레외처리

```
void Rational :: SetDenominator(int denom) {
    if (denom != 0) {
        DenominatorValue = denom;
    }
    else {
        throw(denom);
```

```
    }
}
Rational::Rational(int numer, int denom) {
    SetNumerator(numer);
    try {
        SetDenominator(denom);
    }
    catch ( int d) {
        cerr << "Illegal denominator:" << d
          << "using 1" << endl;
        SetDenominator(1);    }
    }
void Rational:: Extract(istream &sin) {
    int numer ;  int denom ;
    char slash;
    sin >> numer >> slash >> denom;
    SetNumerator(numer);
    try {
        SetDenominator(denom);
    }
    catch ( int d) {
        cerr << "Illegal denominator:" << d << endl;
        cerr << "Reenter rational value:" ;
        Extract(sin);
    }
    return;
      }
```

4.2.2 일반적인 례외처리

C++는 임의의 형태의 례외를 처리할수 있는 특수한 catch처리기를 정의하도록 한다.

```
try {
    cout << "Enter number:"
    double d;
    cin >> d;
    if ( d < 0) {
        throw(d);
    }
    char *p = new char;
    cout << "Enter character:" ;
```

```
        char c;
        cin >> c;
        if (isupper( c )) {
            throw( c);
        }
    }
    catch(bad_alloc b) {
        cout << "Cannot satisfy new request"  << end1;
    }
    catch( )
        cout << "Exception was thrown" << end1;
        throw(c);
    }
```

선행코드토막은 두개의 catch처리기들로 이루어 진 try블로크와 catch처리목록으로 구성되였다. 두번째 catch처리에서는 그의 파라메터목록에 대한 생략부호가 서술되여 있다. 생략부호는 이 catch처리기가 자기의 목록안에서 앞선 처리기들이 처리하지 못하는 례외들을 처리한다는것을 의미한다. catch처리기를 정의하는 순서는 중요하다.

례외는 처리할 때 파라메터목록이 발생한 례외를 만족시키는 첫번째 처리기를 준다. 따라서 생략부호 catch처리기의 정의는 throw명령문을 포함하는것이 중요하다. 례외처리기가 어떤 파라메터목록이 없이 례외를 전송할 때 실지로 접수된 같은 례외를 재전송한다. 이 재전송은 반복적인 호출이 아니다. 그것은 추가적인 처리가 필요하며 현재처리기가 주어 진 처리를 진행할 처리기가 아니라는것을 지적한다. 추가적인 처리는 발생한 례외에 대하여 다른 례외처리기검색을 함수호출로 진행한다.

4.2.3 가능한 례외의 지정

함수의 대면부에서와 같이 례외의 어느 형이 호출한 함수를 복귀할수 있는가를 렬거할수 있다. 이 렬거는 파라메터목록에 따른다. 이 지정은 넘길수 있는 객체의 형을 기입하기 위하여 넘기는 식에 따라 예약어 throw를 구성한다. 아래의 함수 f()에 대한 원형화에서 **int**와 **double**은 f()가 호출될 때 처리를 요구할수 있는 례외의 형이다.

```
    void f(char c)  throw(int, double) ;
```

함수 f()가 **int**나 **double**(혹은 일반적으로 throw목록안의 형들)형이 아닌 례외를 발생시키는 경우 례외는 std이름공간함수 unexpected()의 호출을 기록하게 되는데 그것은 기정으로 프로그람을 끝내게 한다. 실례로 함수 f()가 다음과 같이 정의되였다고 하자.

```
    void f(char c)  throw(int, double) {
        if (isupper ( c))
            throw(1);
        else if (islower( c))
            throw(1.0);
        else if ( c == '.')
```

```
            throw( c);
    }
```

만일 호출 f('a')가 진행되면 double형례외는 f()가 호출되는 함수를 발생한다. 그대신 호출 f('A')가 진행되면 int형례외는 호출함수 f()를 발생한다. 호출 f('.')이 진행되면 프로그람은 완료된다. 그것은 char형례외로 넘어 가기때문에 끌나며 여기서 char형은 int나 double형도 아니며 int나 double형에서 파생되지도 않는다.

4.2.4 동적할당례외의 포착

C++표준을 리용하기에 앞서 그것을 11장에서 론의하였는데 만일 빈 기억기요구가 만족되지 않는다면 new연산은 빈 주소(0)를 돌려 주도록 정의되였다. 또한 C++표준에 대하여 론의하였는데 만일 new요구를 만족시키지 못한다면 례외가 발생된다. 례외처리를 위한 스위치는 빈 기억기요구의 형에 따라 임의로 쓸수 있으며 여러가지 기능을 수행한다. 례외형은 new요구가 bad_alloc라는 불만족한 new요청에 의하여 발생한다.

bad_alloc형은 이름공간의 한 부분으로서 새로운 서고안에서 정의되는데 그것은 표준서고들중의 하나이다(표준원본들은 xalloc형을 리용하며 이것은 례외서고의 부분이다).

아래의 프로그람은 빈 기억기의 해당크기를 결정한다. 새로운 new요청이 발생될수 있는 수를 계산한다. 계수는 bac_alloc가 발생할 때 완료된다.

```
#include <iostream>
#include <string>
#include <new>
using namespace std;
int main() {
    long int size = 0;
    try {
        while (true) {
            char *p = new char;
            ++size;
        }
    }
    catch(bad_alloc b) {
    //아무것도 필요없다.}
    cout << “Free store size:” << size << end1;
    return 0;
}
```

프로그람은 만족된 new요구수를 표현하는 국부객체 size의 정의로부터 시작된다. 객체의 size는 0으로 초기화된다.

try블로크는 그다음에 초기화된다. Try블로크는 례외가 발생할 때까지 반복동작을 수행하는 while 순환으로 구성되여 있다.

```
try {
    while (true) {
        char *p = new char;
        ++size;
    }
}
```

new요구는 while순환문을 매번 실행할 때마다 나타난다. 례외가 발생하지 않았다면 순환의 크기는 증가한다. 례외가 발생되면 그것은 new의 조작으로부터 생겨 나는 bad_alloc례외이다. 례외는 catch(bad_alloc)처리기에 의하여 처리된다. 그 과정이 필요하지 않기때문에 처리기자체는 비게 된다.

```
catch(bad_alloc b) {
    //아무것도 필요없다.
}
```

조종은 처리기에서부터 catch처리기로 발생한 삽입명령문에로 전송된다. 삽입은 빈 기억기의 초기 크기를 표시한다.

```
cout << "Free store size :" <<size <<end1;
```

4.2.5 유산코드

유산코드가 남아 있게 되므로 C++콤파일러는 일반적으로 빈 주소가 불만족한 new조작결과로 돌려지는 간단한 방법을 제공한다. 그 방법은 void함수 Set_new_handler()를 리용하는것이다. 함수 Set_new_handler는 새로운 서고이다. 함수는 new연산자를 위한 례외처리기에 대한 지적자와 같은 파라메터를 요구한다. 빈 주소가 실제파라메터로 리용되면 불만족한 new조작을 위하여 빈 주소를 돌려 주는 고정동작은 일반적으로 효과적이다. 표준적으로 불만족한 new요구에 대한 빈 지적자를 되돌리는 new연산변수를 정의한다. 이 변수는 다음의 코드토막에서 보여 준다.

```
char *p = new(nothrow) char;
if (p) {
    cout << "New request was satisfied"<< end1;
}
else {
    cout << "New request was not satisfied"<< end1;
}
```

객체 nothrow는 새로운 서고에서 정의된 상수이다. 이 상수는 new연산이 진행될 때 연산이 new요구를 만족시킬수 없다면 례외를 발생하지 않는다. 그 대신 빈 주소는 불만족한 new요구에 대하여 돌려 주는것이다.

4.3 Friend속성

정보은페규칙을 비롯하여 대부분의 모든 규칙은 레외를 가진다. 선택된 비성원함수나 클라스로 자료성원들에 접근하는 경우에 C++의 Friend속성이 아주 적합하다. 14장에서 Friend속성을 리용하여 순차목록의 개발을 보여 준다. 그러나 대체로 머리말작성에서 그것을 중요하게 여기지 않으므로 부록에서도 Friend속성을 간단히 론한다. 8장의 Rational ADT의 내용에서도 그러하지만 Friend속성을 리용할수 있는 방법을 간단하고 편리한 실례로 제공한다.

4.3.1 Rational의 다른 실현

Rational산수 및 흐름연산자가 보조연산자에 의하여 실행된다는것은 이미 알고 있다. 연산자들은 사용의 일치성을 위하여 보조적으로 작성된다. 실례로 u가 Rational객체라면 다음과 같은 두개의 명령문은 정확히 실행된다.

```
cout << (u + 2) << end1;
cout << (2 + u) <<end1;
```

연산자들이 합리적인 성원들로 작성되였다면 두번째 삽입은 C++규칙에 따라 콤파일할수 없다. 8장의 보조연산자들은 추가 및 증가연산자들로서 다음과 같다.

```
Rational operator + (const Rational &r, const Rational &s ) {
    return r .Add(s);
}
Rational operator*(const Rational &r, const Rational &s) {
    return r, Multiply(s);
}
```

Rational클라스의 연산자 Friend를 만들어 보자 .

클라스의 Friend는 공개, 비공개 혹은 보호를 고려하지 않고 모든 클라스성원에 접근할수 있다. 다른 함수, 연산자, 클라스에 대한 동료관계를 허락하기 위하여 그 클라스정의에서 함수, 연산자, 클라스의 원형에 대하여 C++변경자 Friend를 리용한다. Friend를 리용한 Rational클라스에 대한 정의를 목록 4-3에 주었다. 산수와 흐름연산자들의 기본형태들은 초기선언요소와 같이 Friend속성과 함께 클라스정의에 있다는것을 관찰한다.

목록 4-3. 동료산수연산자와 흐름연산자에 의한 Rational클라스의 정의

```
class Rational {
    // Friend연산자
    friend Rational operator+(const Rational &r, const Rational &s);
    friend Rational operator*(const Rational &r, const Rational &s);
    friend ostream& operator<<(ostream  &sout, const Rational &s);
    friend istream& l operator>>(istream &sin,  Rational &r,
    public: //성원함수
        //기정구축자
        Rational ();
        //두번째 구축자
```

```
        Rational(int numer, int denom = 1);
      protected:
            //검토자
            int GetNumerator() const;
            int GetDenominator()  const;
            //변이자
            void SetNumerator(int numer);
            void SetDenominator(int denom);
     private:
            //자료성원들
            int NumeratorValue;
            int DenominatorValue;
};
```

Friend함수나 클라스정의는 특별한 문법이 필요없다. 목록 4-4는 비공개부성원들을 리용하는 유리수더하기와 곱하기에 대한 연산자정의를 준다.

목록 4-4. Rational의 동료관계능력을 리용한 보조연산자의 실현부

```
//Rational의 추가
Rational operator+(const Rational &r, const Rational &s) {
    int a = r.GetNumerator();
    int b = r.GetDenominator();
    int c = s.GetNumerator();
    int d = s.GetDenominator();
    return Rational (a*d + b*c, d*b);}  //다중 Rational
Rational operator*(const Rational &r, const Rational &s) {
    int a = r.GetNumerator();
    int b = r.GetDenominator();
    int c = s.GetNumerator();
    int d = s.GetDenominator();
    ` Rational (a*c, d*b);
}
```

목록 4-4에서 두 연산자정의는 류사하다. 국부객체 a, b, c,를 구축하고 연산수 r와 s의 표시자와 이름짓기요소를 표현한다. 연산자들은 Rational클라스의 Friend이기때문에 보호된 성원함수 GetNumerator()와 GetDenominator()에 접근할수 있다. 객체 a, b, c와 d는 그 연산을 위한 값을 되돌려서 실행한 Retional객체를 생성하기 위하여 리용되였다.

Friend수법이 함수, 연산자, 클라스가 어떤 클라스의 자료표현을 조작할수 있게 하지만 정보은페에 관해서는 기본적으로 안전성을 담보할수 없다.

부록 5. EzWindows API참고서

이 부록은 EzWindows API의 형, 클라스, 능력을 개괄한다.

5.1 렬거형

EzWindow의 API는 3개의 렬거형. 즉 color, WindowStatus, BitMapStatus를 정의한다. 렬거형 color는 simpleWindow에 현시할수 있는 가능한 색을 위하여 상징적인 기호이름들을 제공한다.

enum color {Black, white, Red, Green, Blue, Yellow, Cyan, Magenta};

렬거형 WindowStatus는 SimpleWindow객체를 위한 가능한 상태를 정의한다.

enum WindowStatus {WindowClosed, WindowOpen, WindowFailure};

여기서 매 파라메터를 설명한다면 다음과 같다.

- WindowClosed는 열려 지지 않은 창문을 가리킨다. 객체는 이러한 상태를 가진 창문에 현시될수 없다.
- WindowOpen은 열려 진 창문을 가리킨다. 객체는 이러한 상태를 가진 창문에서 현시될수 있다.
- WindowHello 는 실패상태를 가리킨다. 객체는 이러한 상태를 가진 창문에 현시될수 없다.

렬거형 BitMapStatus는 비트매프객체의 가능한 상태를 정의한다.

enum BitMapStatus { NoBitMap, BitMapOkay, NoWindow};

여기서 매 파라메터의 설명은 다음과 같다.

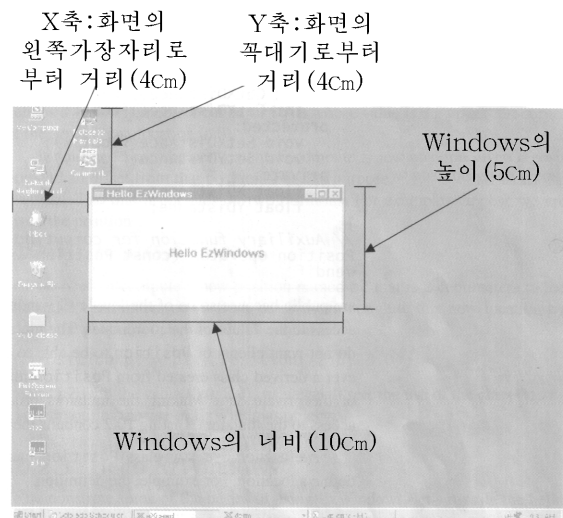
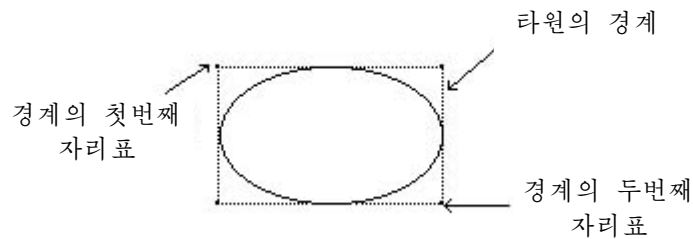
- NOBitMap는 현시할 2진화상이 없다는것을 가리킨다.
- BitMapOkay는 련관된 창문과 현시할 2진화상이 있다는것을 가리킨다.
- NoWindow는 2진화상을 포함하는 련관된 창문이 없다는것을 가리킨다.

5.2 자리표계

그림 5-1은 EzWindows의 자리표계를 보여 준다. 화면의 왼쪽꼭대기가 원점으로 된다. 모든 자리표는 원점에서부터 cm단위로 표현된다. EzWindows객체의 크기의 측정단위는 역시 cm이다.

일부 EzWindows의 API함수들은 객체의 크기를 지정하는 속박통을 리용한다. 실례로 다음의 도표는 타원을 표현하기 위한 속박통을 보여 준다.

그림 5-1. EzWindows의 자리표계

경계는 도형을 포함하는 4각형의 왼쪽꼭대기와 오른

쪽아래점의 자리표에 의하여 결정된다.

5.3 클라스 Position

클라스 Position은 창문객체들의 론리자리표계를 정의하는 객체이다. 이 클라스는 두개의 공개구축자를 제공한다.

Position::Position(**float** x = 0.0, **float** y = 0.0)
Y자리표값으로서 y값, X자리표값으로서 x를 가지는 Position객체를 창조한다.

Position클라스는 두개의 공개성원함수를 제공한다.

int Position::GetXDistance() **const**
X자리표값을 되돌린다.

int Position:: GetYDistance() **const**
Y자리표값을 돌린다.

+연산자는 연산수로서 Position객체를 리용하기 위하여 다중정의된다.

Position **operator** +(**const** Position &a, **const** Position &b)
a와 b의 y자리표값, a 와 b의 자리표값의 합을 표현하는 x자리표값과 y자리표값을 되돌린다.

Position **operator** -(**const** Position &a, **const** Position &b)
a와 b의 y자리표값, a와 b의 자리표값의 차를 표현하는 x자리표값과 y자리표값을 되돌린다.

5.4 클라스 SimpleWindow

클라스 SimpleWindow는 객체가 정의하고 조작할수 있는 간단한 창문현시를 하게 하는 기능을 가진 클라스이다. 이 클라스는 여러개의 공개성원함수를 제공한다.

SimpleWindow :: SimpleWindow(**const char** *WindowTitle = “Untitled”,
float Width = 8.0f ,
float Height = 8.0f ,
const Position &WindowPosn = Position(3.0f, 3.0f));

도형객체를 현시하기 위하여 SimpleWindow를 만든다. 파라메터 WindowTitle은 창문의 제목띠에 현시할 문자렬에 대한 지적자이다. 기정제목은《Untitled》이다. 파라메터 Width는 창문의 넓이이다.

기정너비는 8cm이다. 파라메터 Height는 창문의 높이이다. 기정높이는 8cm이다. 파라메터 WindowPosition은 창문의 위치이다. 첫 자리표는 화면의 왼쪽변두리에서 얼마큼 떨어 졌는가를 나타낸다. 두번째 자리표는 화면의 꼭대기변두리에서 얼마큼 떨어 져 있는가를 나타낸다. 기정위치는 (3.0, 3.0)이다. 이 위치는 화면의 왼쪽변두리에서 3cm, 꼭대기변두리에서 3cm 떨어 졌다는것을 의미한다.

```
SimpleWindow :: SimpleWindow(const string &WindowTitle,
  float Width = 8.0f,  float Height = 8.0f,
  const Position &WindowPosn = Position(3.0f, 3.0f));
```

도형객체를 현시하기 위하여 SimpleWindow를 만든다. 파라메터 WindowTitle은 창문의 제목띠에 현시할 문자렬에 대한 지적자이다. 파라메터 Width는 창문의 넓이이다. 기정너비는 8cm이다. 파라메터 height는 창문의 높이이다. 기정높이는 8cm이다. 파라메터 WindowPosition은 창문의 위치이다. 첫 자리표는 화면의 왼쪽변두리에서 얼마만큼 떨어 졌는가를 나타낸다. 두번째 자리표는 화면의 꼭대기변두리에서 얼마만큼 떨어 져 있는가를 나타낸다. 기정위치는 (3.0, 3.0)이다. 이 위치는 화면의 왼쪽변두리에서 3cm, 꼭대기변두리에서 3cm 떨어 져 있다는것을 의미한다.

SimpleWindow클라스는 여러개의 공개성원함수를 제공한다.

```
WindowStatus SimpleWindow::Close();
```

이 공개성원함수는 창문을 닫고 모든것을 사라지게 한다. 귀환값은 WindowClosed이다.

```
void SimpleWindow::Erase(const Position &UpperLeft,
  float Width, float Height);
```

이 공개성원함수는 4각형지역을 지운다. 4각형의 왼쪽웃구석은 Position UpperLeft에 의하여 지정된다. Width와 height로 설정된 창문령역을 지운다.

```
Position SimpleWindow :: GetCenter()  const;
```

이 공개성원함수는 창문의 중심위치를 얻게 한다. 함수는 창문의 중심에 대한 론리자리표를 표현하는 Position값을 돌려 준다. 또한 창문의 왼쪽과 오른쪽변두리까지의 거리를 측정한다.

```
float SimpleWindow :: GetHeight() const;
```

창문의 높이를 돌려 준다.

```
WindowStatus SimpleWindow :: GetStatus()  const;
```

창문의 상태를 나타내는 WindowStatus값을 돌려 준다.

```
float SimpleWindow :: GetWidth()  const;
```

창문의 너비를 돌려 준다.

```
float simpleWindow :: GetXPosition() const;
```

창문의 X자리표값을 돌려 준다.

```
float SimpleWindow :: GetYPosition() const;
```

창문의 Y자리표값을 돌려 준다.

```
void SimpleWindow :: Message (const string &Msg = Message) ;
```

통보를 가진 경보창문을 내보낸다. 파라메터 MSG는 경계창문에 현시하여야 할 문자이다.

WindowStatus SimpleWindow :: Open() ;

화면에 창문을 현시하고 객체를 현시한다. 함수는 창문의 상태를 나타내는 WindowStatus 값을 돌려 준다.

void SimpleWindow :: RenderEllipse(
const Position &UpperLeft,
const Position &LowerRight, const color &c,
const bool Border = false);

타원을 그린다. 경계는 UpperLeft와 LowerRight에 의하여 그 크기가 결정된다. 타원은 빨간색으로 안에 색칠된다. 만일 Border가 거짓이면 경계 없는 타원을 그린다. 이 값이 참이라면 새까만 경계를 가진 타원을 그린다.

void SimpleWindow :: RenderPolygon(
const vector<Position> &PolyPoints, int Npoints,
const color &c, const bool Border = false);

닫긴 다각형을 그린다. 다각형의 매 점들은 PolyPoints벡토르에 의하여 결정된다. 파라메터 nPoint는 다각형의 점의 개수이다. 다각형은 빨간색으로 색칠된다. Border가 거짓이라면 경계 없는 다각형을 그리며 참이라면 새까만 경계를 가진 도형을 그린다.

void SimpleWindow::RenderPolygon(
const Position PolyPoints[ ], int nPoints,
const color &c, const bool Border = false););

닫긴 다각형을 그린다. 다각형의 매점들은 PolyPoints벡토르에 의하여 결정된다. 파라메터 nPoint는 다각형의 점의 개수이다. 다각형은 빨간색으로 색칠된다. Border가 거짓이라면 경계 없는 다각형을 그리며 참이라면 새까만 경계를 가진 도형을 그린다.

void SimpleWindow::RenderText(
const Position &UpperLeft,
const Position &LowerRight,
const String &Msg = "Message",
const color &textcolor = Black,
const color &BackGroundcolor = White);

창문에 본문문자렬을 현시한다. 파라메터 UpperLeft는 본문통보창의 왼쪽웃구석위치이다. 파라메터 LowerRight는 본문통보창의 오른쪽아래구석의 위치이다. 파라메터 MSG는 창문에 현시되는 문자렬이다. 기정으로 설정되는 통지문은 "Message"이다. 파라메터 Textcolor는 본문통보문의 색이다. 기정으로 설정된 본문색은 검은색이다. Backgroundcolor파라메터는 본문의 배경색이다. 기정설정값은 흰색이다.

void SimpleWindow::RenderText(
const Position &UpperLeft,
const Position &LowerRight,

const String &Msg = "Message",
const color &textcolor = Black,
const color &BackGroundcolor = White);

창문에 본문문자렬을 현시한다. 파라메터 UpperLeft는 본문통보창의 왼쪽웃구석위치이다. 파라메터 LowerRight는 본문통보창의 오른쪽아래구석의 위치이다. 파라메터 MSG는 창문에 현시되는 문자렬이다. 기정으로 설정되는 통보문은 "Message"이다. 파라메터 Textcolor는 본문통보문의 색이다. 기정으로 설정된 본문색은 검은색이다. Backgroundcolor파라메터는 본문의 배경색이다. 기정설정값은 흰색이다.

void SimpleWindow::SetMouseClickCallback(
MouseClickCallbackFunction f);

마우스누르기에 대한 역호출을 등록한다. 마우스누르기사건이 창문에서 발생하면 함수 f()가 호출된다. 이 함수는 단일한 const Position &파라메터형으로 선언되여야 하며 되돌림형은 int이다. 이 함수의 귀환값은 사건이 성과적으로 처리되였는가 , 아닌가를 가리킨다. 값 1은 성공, 0은 오유가 발생하였다는것을 의미한다.

void SimpleWindow::SetRefreshCallback(
RefreshCallbackFunction f);

새로운 통지문에 대한 역호출을 등록한다. 창문이 새로운 사건을 접수하면 함수f()가 호출된다. 함수 f()는 파라메터가 없이 선언되여야 하며 int값을 되돌려 주어야 한다. 이 함수의 귀환값은 사건이 성과적으로 처리되였는가, 그렇지 않은가를 나타낸다. 값이 1이면 성공, 0이면 오유발생을 의미한다.

void SimpleWindow::SetQuitCallback (QuitCallbackFunction f);

완료통지문을 위한 역호출을 등록한다. 함수 f()는 창문이 완료사건을 접수할 때 호출된다. 이 함수는 파라메터가 없이 호출되며 int값을 되돌린다. 이 함수의 귀환값은 사건이 성공적으로 처리되였는가 아닌가를 나타낸다. 값이 1이면 성공, 0이면 오유발생을 의미한다.

bool SimpleWindow::;StartTimer(int interval);

박자계수기의 가동을 시작한다. interval파라메터는 박자계수기사건들에서 미리초를 단위로 하는 수자이다. 귀환값은 박자계수기가 성공적으로 가동하였는가 , 아닌가를 가리킨다. 귀환값이 <참>이면 성공, <거짓>이면 박자계수기가 설정될수 없다는것을 의미한다.

void SimpleWindow::StopTimer();

박자계수기를 정지한다.

void SimpleWindow::SetTimerCallback(
TimerTickCallbackFunction f);

박자계수기의 박자를 위한 역호출을 등록한다. 함수 f()는 박자계수기의 박자가 발생할 때 호출된다. 함수 f()는 파라메터가 없이 선언되며 되돌림값은 int형이다. 이 함수의 귀환값은 사건이 성공적인가, 아닌가를 가리킨다. 값이 1이면 성공, 0이면 오유발생을 의미한다.

5.5 WindowObject클라스

이 클라스는 Shape클라스의 기초클라스이다. 클라스는 공개구축자를 제공한다.

Windowobject ::Windowobject(SimpleWindow &w,**const** Position &p);

창문 w에서 중심이 p인 windowObject를 창조한다.

WindowObject클라스는 여러개의 공개성원함수들을 제공한다.

Position WindowObject::GetPosition() **const**;

창문객체의 위치를 돌려 준다.

void WindowObject::GetPosition(**float** &Xcoord, **float** &Ycoord) **const**;

창문객체의 위치를 돌려 준다. X자리표는 Xcoord에 , Y자리표는 Ycoord에 돌려 주게 된다.

SimpleWindow& WindowObject ::Getwindow() **const**;

WindowObject를 포함한 창문을 돌려 준다.

void WindowObject::SetPosition(**const** Position &p);

Windowobject의 위치를 p에 설정한다.

void WindowObject::SetPosition(**float** Xcoord, **float** Ycoord);

WindowObject의 위치를 Position(Xcoord, Ycoord)에 설정한다.

5.6 RaySegment클라스

이 클라스는 SimpleWindow도형체계에서 선들을 표시한다. 이 클라스는 WindowObject클라스로부터 파생된다. 선은 시작점을 가지며 그 점은 Position이다. 이 자료성원은 WindowObject로부터 계승된다. 이 클라스는 아래에 선언된 두개의 공개구축자를 제공한다.

RaySegment::RaySegment(SimpleWindow &w,
const Position &StartPoint, **const** Position &EndPoint,
const color &c = Black, **float** Thickness = 0.1f,
bool Arrowhead = false);

선을 표시하기 위하여 RaySegment객체를 창조한다. 선언 SimpleWindow w.에 포함된다. 선의 시작점은 StartPoint이며 끝점은 EndPoint이다. 선은 색값 C를 가지는데 그 값은 검은색으로 기정으로 설정되여 있다. 선언은 Thickness파라메터를 가지는데 이것은 선의 두께를 표시한다. 기정설정두께는 0.1cm이다. 만일 Arrowhead가 참이면 선의 끝점에 화살촉이 그려 진다. 다른 경우에는 선에 화살촉이 없다. 기정설정형태는 화살촉이 없다.

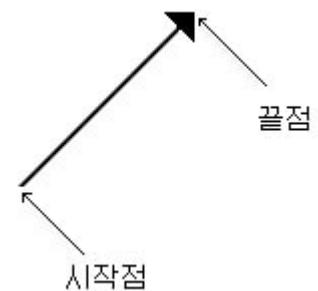



그림 5-2. EzWindows

RaySegment::RaySegment(SimpleWindow &w, **float** StartX,
float StartY, **float** EndX, **float** EndY,
const color &c = Black, **float** Thickness = 0.1f,
bool Arrowhead = false);

선을 표시하기 위하여 RaySegment객체를 창조한다. 선은 SimpleWindow w에서 포함된다. 선의 시작점은 Position (startX, startY)이며 끝점은 Position(EndX, EndY)이다. 선은 색값 C를 가지는데 기정설정값은 검은색이다. 선은 Thickness파라메터를 가지는데 cm로 표현된다. 기정설정두께는 0.1cm이다. 만일 Arrowhead가 참이면 선의 끝쪽에 화살촉이 그려 진다. 그밖의 경우에는 화살촉이 그려 지지 않는다. 기정으로 설정한 선의 형태는 화살촉이 없는것이다.

Raysegment클라스는 아래에 선언된 여러개의 공개성원함수들을 제공한다.

void RaySegment::ClearArrowhead();

화살촉이 없는 선이 그려 지게 한다.

void RaySegment::Draw();

창문에 선을 그린다.

void RaySegment::Erase();

선을 지운다.

color RaySegment::Getcolor() **const**;

선의 색을 돌려 준다.

Position RaySegment::GetEndPoint() **const**;

선의 끝점을 얻는다.

void RaySegment::GetEndPoint(**float** &x, **float** &y) **const**;

선의 끝점을 얻는다.

float RaySegment::GetLength() **const**;

선의 길이를 계산하여 cm단위로 돌려 준다.

void RaySegment::;GetPoints(Position &Start, Position &End) **const**;

선의 시작점과 끝점을 얻는다

Position RaySegment::GetStartPoint() **const**;

선의 시작점을 얻는다.

void RaySegment::GetStartPoint(flost &x, **float** &y) **const**;

선의 시작점을 얻는다.

float RaySegment::GetThickness() **const**;

선의 두께값을 얻는다.

bool RaySegment :: HasArrowhead();

선이 화살을 가지면 참, 아니면 거짓을 돌린다.

void RaySegment ::SetArrowhead();

선이 화살촉을 가지도록 설정한다.

void RaySegment::Setcolor(**const** color &c);

선의 색을 c값으로 설정한다.

void RaySegment ::SetEndPoint(**const** Position &p);

선의 끌점을 자리표 p에 설정한다.

void RaySegment:: SetEndPoint(**float** x, **float** y);

선의 끌점위치를 position(x, y)에 설정한다.

void RaySegment :: SetPoints(**const** Position &StartPoint, **const** Position &EndPoint);

선의 시작점과 끌점을 설정한다.

void RaySegment :: SetStartPoint(**const** Position &p);

선의 시작점을 p에 설정한다.

void RaySegment :: SetStartPoint(**float** x, **float** y);

선의 시작점을 (x, y)에 설정한다.

void RaySegment :: SetThickness(**float** t);

선의 두께를 설정한다.

5.7 Shape클라스

이 클라스는 CircleShape, EllipseShape, RectangleShape, TriangleShape, SquareShape클라스들의 기초클라스이다. 클라스는 아래에 서술된 공개구축자를 제공한다.

Shape ::Shape(SimpleWindow &w, **const** Position &p,
const color &c= Red);

창문 w의 p위치에 중심을 두고 있는 Shape객체를 창조한다. 객체의 색은 C로 설정하며 그 값은 기정설정으로 Red이다.

Shape클라스는 아래에 선언한 여러개의 공개성원함수를 제공한다.

void Shape::ClearBorder()

테두리를 지운다.

virtual **void** Shape::Draw() = 0;

성원함수 Draw()는 순수가상함수이다.

color shape::Getcolor() **const**;

객체의 색을 얻는다.

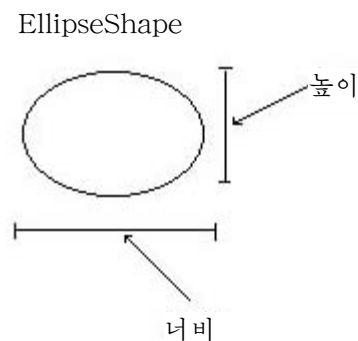
bool Shape::HasBorder() **const**;

형태가 테두리를 가지면 참을, 아니면 거짓을 돌려 준다.

void Shape::SetBorder();
  테두리를 설정한다.

void Shape::Setcolor(const color &c);
  객체의 색을 C로 설정한다.

5.8 EllipseShape클라스

이 클라스는 Shape클라스에서 공개적으로 파생된다. EzWindows EllipseShape를 아래에 보여 준다.

EllipseShape클라스는 다음과 같은 공개구축자를 가진다.

EllipseShape :: EllipseShape(Simplewindow &w,
 const Position &p, const color &c = Red,
 float Width = 1.0f, float Height = 2.0f);
  타원을 표현하기 위하여 EllipseShape를 창조한다. 타원은 창문 w 의 p에 중심을 두었다. 타원의 색값은 C이며 그 기정값은 Red이다. 타원은 너비 Width와 높이 Height를 가진다. 파라메터 Width 와 Height의 기정값은 각각 1.0, 2.0이다. 이 파라메터들은 cm단위이다.

EllipseShape클라스는 다음의 공개성원함수를 가진다.

void ellipseShape :: Draw();
  창문에 타원을 그린다.

void ellipseshape::Erase();
  타원을 지운다.

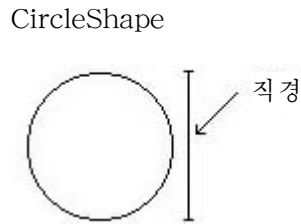
float EllipseShape:;GetHeight() const;
  객체의 높이를 얻는다(cm).

void ellipseShape::GetSize(float &Width, float &Height) const;
  객체의 너비, 높이를 얻는다(cm).

void EllipseShape:: GetWidth() const;
  객체의 너비를 얻는다(cm).

void ellipseshape::SetSize(float Width, float Height);
  타원의 너비를 Width로, 높이를 Height로 설정한다(cm).

5.9 CircleShape클라스

이 클라스는 Shape클라스에서 공개적으로 파생된다. EzWindows CircleShape를 아래에 보여 준다.

CircleShape클라스는 다음과 같은 공개구축자를 가진다.

```
CircleShape::CircleShape(SimpleWindow &w,
  const Position *p , const color &c = Red ,
  float Diameter = 1.0f);
```

원을 표현하기 위하여 CircleShape객체를 창조한다. 원은 창문 w의 p위치에 중심을 두었다. 원은 색갈 C를 가지며 기정값은 Red이다. 직경(Diameter)도 가진다. 이 파라메터의 기정값은 1.0이다. 이 파라메터는 cm단위이다.

CircleShape클라스는 다음의 공개성원함수를 가진다.

```
void CircleShape::Draw();
```

창문에 원을 그린다.

```
void CircleShape::Erase();
```

원을 지운다.

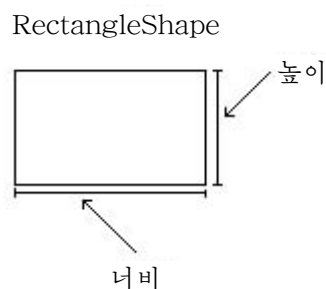
```
float CircleShape::GetDiameter() const;
```

원의 직경을 얻는다.

```
void CircleShape :: SetSize(float Diameter);
```

원의 직경을 Diameter로 설정한다.파라메터 Diameter는 cm단위이다.

5.10 RectangleShape클라스

이 클라스는 Shape클라스에서 공개적으로 파생된다. Ezwindows의 RectangleShape는 아래에서 보여 준다.

RectangleShape

RectangleShape클라스는 다음의 공개함수를 가진다.

RectangleShape :: RectangleShape(SimpleWindow &w,
  const Position &p, **const** color &c = Red,
  float width = 1.0f, **float** Height = 2.0f);

4각형을 표현하는 RectangleShape객체를 창조한다. 자리표가 p인 위치에 창문 w가 중심을 두었다. 4각형은 색값 C를 가지는데 기정값은 Red이다. 4각형의 너비는 Width이다. 높이는 Height이다. 이 파라메터들의 기정값은 각각 1.0, 2.0이다. 이 파라메터는 cm단위이다.

RectangleShape::RectangleShape(SimpleWindow &w,
  float Xcoord, **float** Ycoord, **const** color &c = Red,
  float Width = 1.0f, **float** Height = 2.0f);

4각형을 표현하는 RectangleShape객체를 창조한다. 4각형은 창문 w의 position(XCoord, Ycoord)에 중심을 둔다. 4각형은 색값 C를 가지는데 기정값은 Red이다. 4각형의 너비는 Width이다. 높이는 Height이다. 이 파라메터들의 기정값은 각각 1.0, 2.0이다. 이 파라메터는 cm단위이다.

클라스 RectangleShape는 다음의 공개성원함수를 가진다.

void RectangleShape::Draw();

창문에 4각형을 그린다.

void RectangleShape::Erase();

4각형을 지운다.

float RectangleShape::GetHeight() **const**;

4각형의 높이를 얻는다.

void RectangleShape::GetSize(**float** &Width,
float &Height) **const** ;

4각형의 너비와 높이를 얻는다.

float RectangleShape::GetWidth() **const**;

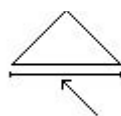
4각형의 너비를 얻는다.

void RectangleShape:: SetSize(**float** width, **float** Height);

4각형의 너비와 높이를 설정한다.

5.11 TriangleShape클라스

이 클라스는 Shape클라스에서 공개적으로 파생된다. Ezwindows의 TriangleShape를 아래에 보여 준다.

TriangleShape

변의 길이

이 클라스는 다음의 공개구축자를 가진다.

```
TriangleShape::TriangleShape(Simplewindow &w,
  const Position &p,  const color  &c = Red,
  float SideLength = 1.0f);
```

바른3각형을 표현하는 TriangleShape객체를 창조한다. 3각형은 창문 w의 p위치에 중심을 둔다. 3각형은 색값 C를 가지는데 기정값은 Red이다. 3각형의 한변의 길이는 SideLength이다. 이 값은 기정으로 1.0이다. 단위는 cm이다.

클라스 TriangleShape는 다음의 공개성원함수를 가진다.

```
void TriangleShape::Draw();
```

3각형을 그린다.

```
void TriangleShape::Erase();
```

3각형을 지운다.

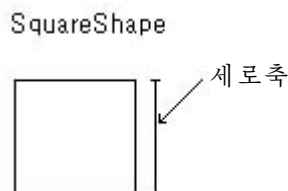
```
float TriangleShape:: GetsideLength()  const;
```

3각형의 변의 길이를 얻는다.

```
void TirangleShape:: SetSize(float SideLength);
```

3각형의 변의 길이를 설정한다.

5.12 SquareShape클라스

이 클라스는 Shape클라스에서 공개적으로 파생된다. EzWindows의 SquareShape를 아래에 보여준다.

SquareShape클라스는 다음의 공개성원함수를 가진다.

```
SquareShape::;SquareShape(SimpleWindow &Window,
  const Position &Center,  const color &c = Red,
  float  Side = 1.0f);
```

바른4각형을 표시하는 SquareShape객체를 창조한다. 창문 w에서 p위치에 4각형의 중심이 놓인다. 이 4각형은 색값 C를 가지는데 기정값은 Red이다. 매변의 길이는 SideLength이다. 기정값은 1.0이다. 단위는 cm이다.

클라스 SquareShape는 다음의 공개성원함수를 가진다.

```
void SquareShape::Draw();
```

바른4각형을 그린다.

void SquareShape::Erase();

바른4각형을 지운다.

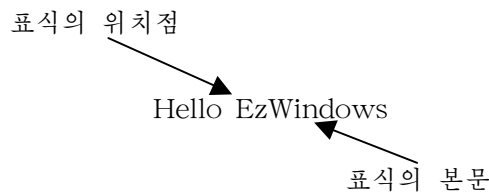
float SquareShape::GetSideLength() **const**;

바른4각형의 매변의 길이를 얻는다.

void SquareShape::SetSize(**float** SideLength);

바른4각형의 변의 길이를 설정한다.

5.13 Lable클라스

이 클라스는 Window객체에서 공개적으로 파생된다. EzWindows 의 Label을 보여 준다.

Label클라스는 다음의 공개구축자를 가진다.

Label1::Label(Simplewindow &w, **const** Position &p,
const string &Text, **const** color &Textcolor = Black,
const color &BackGroundcolor = White);

본문통보문을 표현하는 Label객체를 창조한다. 통보문은 String객체 Text에 포함된다. 통보문은 창문 W의 p위치에 중심을 둔다. 통보문의 색은 Textcolor이다. 기정색은 검은색이다. 통보문은 배경색 BackGroundcolor를 가지며 기정색은 흰색이다.

Label1:: Label1(SimpleWindow &w, **float** Xcoord,
float Ycoord, **const** string &Text,
const color &Textcolor = Black,
const color &BackGroundcolor = White);

본문통보문을 표현하는 Label객체를 창조한다. 통보문은 String객체 Text에 포함된다. 통보문은 창문 W의 Position (Xcoord, Ycoord)에 중심을 둔다.통보문의 색은 Textcolor이다. 통보문의 기정색은 검은색이다. 통보문은 배경색 BackGroundcolor를 가지는데 기정값은 흰색이다.

Label1::Label(SimpleWindow &w, **const** Position &p,
const char "Text, **const** color &Textcolor = Black,
const color &BackGroundcolor = White);

본문통보문을 표현하는 Label객체를 창조한다. Char지적자 Text는 현시할 본문통보문에 대한 지적자이다. 통보문은 창문 w의 p위치에 중심을 둔다. 통보문색은 Textcolor이다. 기정색은 검은색이다. 통보문은 배경색 BackGroundcolor를 가지는데 기정색은 흰색이다.

Label::Label(SimpleWindow &w, **float** Xcoord,

float Ycoord, **const char** *Text,
const color &Textcolor = Black,
const color &BackGroundcolor = White);

본문통보문을 표현하는 Label객체를 창조한다. Char지적자 Text는 현시하는 본문통보문에 대한 지적자이다. 통보문은 창문 w의 Position(Xcoord, Ycoord)에 중심을 둔다.통보문색은 Textcolor이다. 기정색은 검은색이다. 통보문은 BackGroundcolor배경색을 가지며 기정색은 흰색이다.

Label클라스는 다음의 공개성원함수를 가진다.

void Label::Draw();

창문에 표식을 붙인다.

void Label::Erase();

창문에서 표식을 지운다.

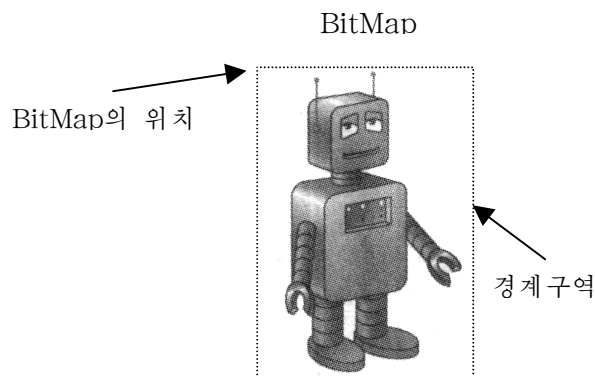
color Label::Getcolor() **const**;

표식의 배경색을 얻는다.

void Label::Setcolor(**const** color &c);

표식의 배경색을 C로 설정한다.

5.14 BitMap클라스

창문형태의 객체(RectangleShape, EllipseShape, CircleShape)들과는 달리 BitMap는 점들의 왼쪽웃구석을 리용하여 자리가 설정된다. EzWindows BitMap를 아래에 보여 준다

BitMap클라스는 다음의 공개구축자를 가진다.

BitMap::BitMap();

BitMapStatus NoBitMap로 BitMap객체를 창조한다. 객체는 임의의 창문과 련관되여 있지 않다.

BitMap::BitMap(SimpleWindow &w);

BitMapStatus NoBitMap로 BitMap객체를 창조한다. 객체는 창문 W와 련관되여 있다.

BitMap::BitMap(Simplewindow *w);

BitMapStatus NoBitMap로 BitMap객체를 창조한다. 객체는 w로 지시한 창문과 련관되여 있다.

클라스 BitMap는 다음의 공개성원함수를 가진다.

bool BitMap::Draw();

련관된 창문에 비트매프객체를 현시한다. 객체의 BitMapStatus는 성공적인 현시를 위해 BitMapOkay로 되여야 한다. 만일 비트매프가 현시되면 함수는 참을 돌려 주지만 다른 경우는 거짓을 돌려 준다.

bool BitMap::Erase();

같은 크기의 흰 4각형을 그려서 화면에 비트매프를 덧쓰기한다.

bool BitMap::IsInside(const Position &p) const;

함수는 p위치가 비트매프안에 놓이면 참, 아니면 거짓을 돌려 준다.

float BitMap::GetHeight() const;

비트매프의 높이를 cm로 돌려 준다.

Position BitMap::GetPosition() const;

비트매프의 위치를 얻는다.

void BitMap::GetSize(float &width, float &Height) const;

비트매프의 너비와 높이를 둘 다 얻는다.

BitMapStatus BitMap::GetStatus() const;

객체와 관련된 BitMapStatus의 현재값을 돌려 준다.

float BitMap::GetWidth() const;

비트매프의 너비를 얻는다.

float BitMap::GetXPosition() const;

비트배렬의 왼쪽웃구석에서부터 련관된 창문의 왼쪽경계선까지 의 거리를 얻는다.

float BitMap::GetYPosition() const;

비트매프의 왼쪽웃구석에서부터 련관된 창문의 웃경계선까지의 거리를 얻는다.

BitMapStatus BitMap::Load(const string &Filename);

비트매프를 설정하기 위하여 이름이 Filename인 파일을 리용한다. 만일 파일이 정확한 비트매프를 포함한다면 객체의 상태는 BitMapOkay로 설정되며 다른 경우에는 NoBitMap로 설정된다.

BitMapStatus BitMap::Load(const char *Filename);

비트매프를 설정하기 위해 문자렬 FileName에 의하여 지적된 이름을 가진 파일을 리용한다. 파일이 정확한 비트매프를 포함하면 객체의 상태는 BitMapOkay로, 그렇지 않으면 그 상태가 NoBitMap로 설정된다.

void BitMap::SetPosition(**const** Position &p);

비트매프의 위치를 p에 설정한다.

void BitMap::SetWindow(SimpleWindow &w);

비트매프는 창문 w와 련관된다. 비트매프의 BitMapStatus가 NoBitMap로 설정된다.

5.15 RandomInt클라스

이 클라스는 지정한 범위내에서 일정한 란수를 만드는 기능을 제공한다. 클라스는 다음의 공개구축자를 가진다.

RandomInt::RandomInt(**int** a = 0, **int** b = RAND_MAX);

RandomInt객체를 창조하는데 그것은 구간 (a, b)에서 모조란수를 발생한다. 기정으로 설정된 구간은 (0, RAND_MAX)이다. RAND _ MAX값은 Stdlib.h에 정의되여있다.

RandomInt::RandomInt(**int** a, **int** b, unsigned **int** Seed);

전체 구간 (a, b)에서 모조란수를 발생하는 RandomInt객체를 창조한다. 모조란수발생기는 Seed에 표시된 값으로 초기화된다.

클라스 RandomInt는 다음의 공개성원함수를 가진다.

int RandomInt::Draw();

다음의 모조란수를 되돌린다.

Unsigned **int** EzRandomize();

모조란수발생기에 체계값을 설정한다.

int RandomInt::GetLow() **const**;

시간간격의 낮은 한계를 되돌린다.

int RandomInt::GetHigh() **const**;

시간간격의 제일 높은 한계를 되돌린다.

void RandomInt::Set**int**erval(**int** a, **int** b);

RandomInt객체를 위한 전체 구간을 (a, b)으로 설정한다.

void RandomInt::SetSeed(unsigned **int** Seed);

모조란수발생기를 체계값으로 설정한다.

5.16 여러가지 함수들

Long GetMilliseconds()

박자를 발생하는 박자계수기값을 돌려 준다. 박자계수기의 박자단위는 ms이다.

void Terminate()

완료통보문을 EzWindows창문관리자에 보낸다.

부록 6. 대상과제파일과 제작파일

API서고들과 여러 실행시간서고들을 리용하는 쏘프트웨어작성은 프로그람작성자에 대해 효과적인 관리부담을 준다. 프로그람작성자는 프로그람작성모듈들이 번역될수 있도록 정확한 머리부정의파일들을 할당해야 한다. 또한 프로그람작성자는 실행파일을 작성하기 위해 그것들이 응용프로그람과 련결될수 있도록 요구되는 서고파일들을 할당해 주어야 한다. 작은 프로그람일지라도 실행파일을 만들기 위해서는 응용프로그람코드와 련결된 여러개의 서고들을 가져야 한다. 쏘프트웨어구축처리를 간단하게 하기 위하여 Borland C++와 Microsoft Visual C++와 같은 개인용콤퓨터번역체계들은 창조, 편집, 번역, 련결 및 오유수정프로그람들을 지원하는 통합개발환경(IDE)를 제공한다. UNIX상에서 통합개발환경을 제공하기보다 지정한 쏘프트웨어가 매 쏘프트웨어개발과제 즉 프로그람창조 및 편집을 위한 편집프로그람, 번역을 위한 번역프로그람, 련결을 위한 련결프로그람, 오유수정을 위한 오유수정프로그람들을 지원하는것이 더 낫다. 번역과 련결과제를 관리하기 위하여 UNIX체계들은 제작(make)파일이라는 프로그람을 제공한다. 이 부록에서는 Ezwindows응용프로그람들을 창조하기 위해 두개의 통속적인 IDE들과 UNIX제작편의프로그람리용법에 대하여 서술한다.

6.1 대상과제와 제작파일의 기초

대상과제(project)파일이나 제작파일(makefile)들은 응용프로그람을 콤파일하고 련결하는 방법을 규정하는 정보들을 저장하고 있다. 대상과제나 제작파일들은 응용프로그람에 대한 원천파일들의 위치, 응용프로그람을 번역하는데 필요한 머리부파일정의(체계와 사용자머리부파일정의)들의 위치 및 응용프로그람을 련결하기 위한 서고들의 위치들과 번역 및 련결하는것의 금지, 허용되는 임의의 콤파일러선택들을 포함한다. EzWindows응용프로그람을 건설하는 대상과제를 창조하는 방법을 보여 주기 위하여 C++와 Microsoft Visual C++IDE들을 리용하여 대상과제파일을 창조한다. 콤퓨터의 구성에 호환시키기 위하여 그것을 변경하는 대상과제에 같은 방법들이 적용될수 있다. UNIX제작파일의 편리를 위해 현존제작파일을 분석하고 또 다른 응용프로그람을 만드는데 리용할수 있도록 변화과정을 서술한다.

이렇게 표현된 제작파일은 여러개의 응용프로그람을 위한 제작파일을 창조하는 모델로서 리용될수 있다. IDE들중 어느 하나를 리용하여 대상과제를 창조하는 단계를 서술할 때 어느 차림표항목을 선택하겠는가를 지적하기 위하여 다음과 같은 표시법을 리용한다. File -> New명령은 다음의 동작을 수행한다. File차림표를 선택하고 여기서 표시된 보조차림표의 New명령을 선택한다. 이런 지령들은 보조차림표의 여러가지 표식들을 지정하기 위해 독립적으로 련결될수 있다. 실례로 File -> New ->Project지령은 New보조차림표가 나타나면 Project지령을 선택해야 한다는것을 지정한다.

다음부분에서는 작업실례프로그람으로서 10장에서 언급한 Simon프로그람을 리용한다. 이 프로그람이 여러개의 원천모듈들을 포함하고 있다는것을 상기시킨다. 그림 6-1은 실행할수 있는 Simon프로그람을 창조하기 위하여 번역되여야 하는 응용프로그람모듈들과 거기에 련결되여야 할 서고파일들을 보여 준다. 6.2부분은 Borland C++ 5.0으로 대상과제를 창조하는 방법을 서술한다. 6.3부분은 Microsoft Visual C++ 5.0으로 대사과제를 창조하는 방법을 서술한다. 이 설명들은 콤파일러에 대한 초기판본으로써 리용자들에게 도움을 준다. 6.4부분은 대표적인 UNIX제작파일과 각이한 콤퓨터의 구성에 호환되도록 그것을 변경하는 방법, 새 응용프로그람을 건설하기 위해 그것을 변화시키는 방법을 서술함으로써 이

부록을 끝마친다.

6.2 Borland C++통합개발환경

이 통합개발환경은 대상과제파일에서 응용프로그람을 건설하는 방법에 대한 정보를 제공한다. Borland대상과제파일들은 확장자 .ide를 가진다. Borland의 통합개발환경그림창문을 그림 6-2에 보여준다. 창문의 제일 꼭대기에 File, Edit, Search 등의 지령이 있다. 지령항목을 눌러 차림표가 나타나게 한다.

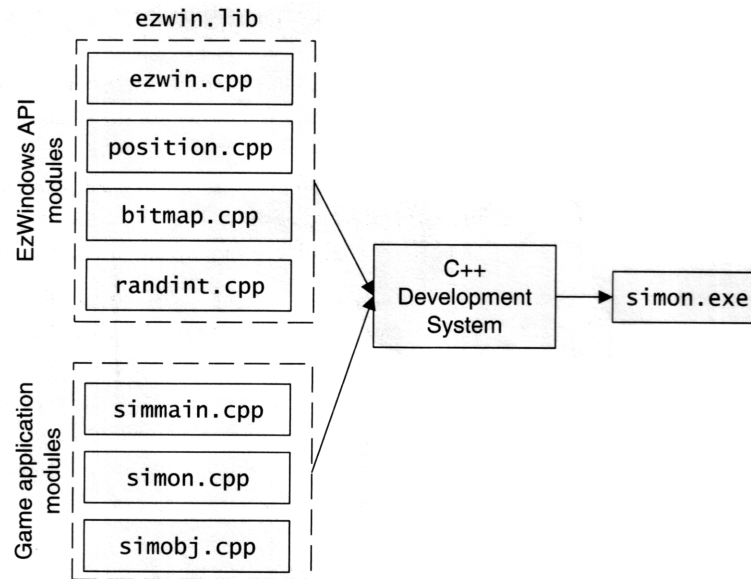
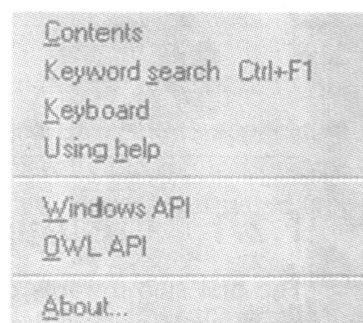



그림 6-1. Simon의 모듈구조

실레로 Help에 대한 누르기는 다음의 보조차림표를 나타낸다. 차림표항목아래에 여러가지 단추들이 있다. 이 령역을 도구띠라고 한다.

도구띠는 자주 리용되는 여러개의 지령건들을 포함한다. 례를 들어 왼쪽에서 첫번째 단추는 파일열기지령을 실행한다. 이 지령은 또한 다음의 동작을 수행하여 실행될수 있다.

File -> Open

Borland C++ IDE 의 기초를 깊이 학습한후 이 환경의 일부 갱신된 특성들을 리용할수 있다. 도구띠의 방조단추를 누르고 요구하는 항목을 눌러 통합개발환경의 임의의 특성에 대한 방조를 얻을수 있다. 특성을 해설하는 방조파일이 현시된다.

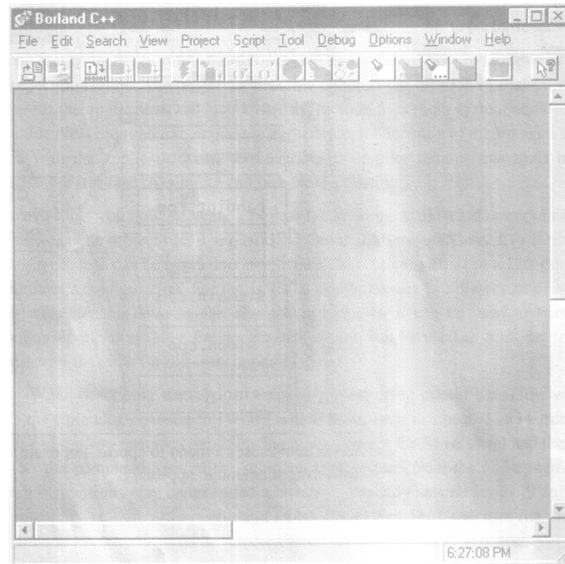



그림 6-2. Borland C++통합개발환경

6.2.1 대상과제파일의 창조

EzWindows응용프로그람을 위한 대상과제를 창조하는 첫 단계는 대상과제파일을 창조하는것이다. 이를 위해 다음의 동작들을 수행해야 한다.

File -> New -> Project

이 차림표선택묶음은 그림 6-3에서 보여 주는 대화칸을 연다. 여러개의 마당들이 여기에 있다. Project Path and Name마당은 대상과제파일을 보관할 위치와 대상과제파일의 이름을 포함하고 있다. 그 위치는 대표적으로 응용프로그람의 원천파일들을 포함한 등록부의 경로로 된다. 실례로 D :\ jwd \ simon 대상과제파일을 지정하려고 한다. 대상과제이름은 대상과제정보를 저장하기 위한 파일의 이름이다. 실례로 Simon.ide안에 대상과제정보를 저장하려고 한다. 따라서 이름마당에 D:\ jwd\ simon\ simon.ide 를 입력한다. 그다음 작성하려고 하는 파일의 이름을 작성해야 한다. TargetName 마당은 실행용파일을 저장하기 위한 파일의 이름을 포함한다. 통합개발환경은 TargetName 마당에 Proj0000이름을 기정으로 설정한다.

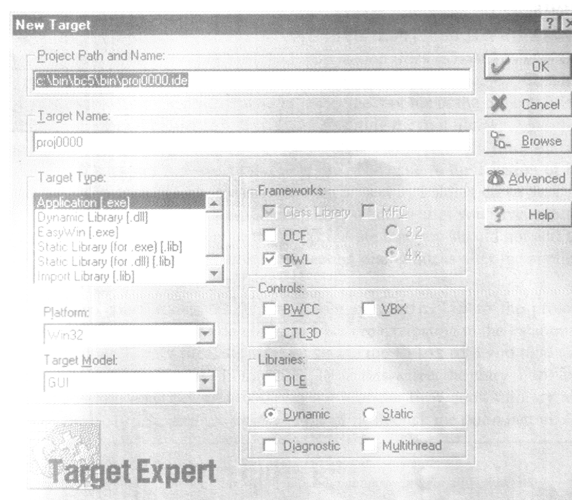



그림 6-3. Borland C++ New Target대화칸

Simon이 호출되도록 건설된 실행파일이 요구된다. 그래서 그 마당의 끝을 누른후 그우에서 후퇴건(Backspace)을 눌러 Proj0000을 지우고 Simon을 입력한다. EzWindows응용프로그람의 목적형태는 Application(.exe)이다. 기정설정에 의하여 이 입력값은 대체로 TargetType흐름띠내림펼침목록으로 선택된 응용프로그람형태이다. 그러나 목적형태 Application이 선택되지 않는다면 리용자가 그것을 선택해야 한다. 가동환경의 적당한 입력값은 win32이다. Target Model마당에 대한 선택은 응용프로그람이 표준입출력장치를 요구하는가, 안하는가에 관계된다. 입출력흐름객체 cin을 리용하여 사용자로부터 건반입력을 접수하거나 cout로 현시하는 EzWindows응용프로그람은 표준입출력장치를 요구한다. 이 EzWindows프로그람에 대한 적합한 선택은 Console이다. 응용프로그람의 입출력수속흐름을 리용하지 않는다면 적합한 선택은 GUI(도형사용자대면부)이다. Simon프로그람은 입출력흐름서고를 리용하지 않는다. 따라서 GUI를 선택해야 한다. 대화칸의 FrameWorks부분에 선택될 항목은 Static뿐이다. 이것이 New Target대화칸의 구성을 완성한다. 완성된 대화칸의 구성을 그림 6-4 에 보여 주었는데 대상과제파일을 기억하기 위해 OK단추를 누른다.

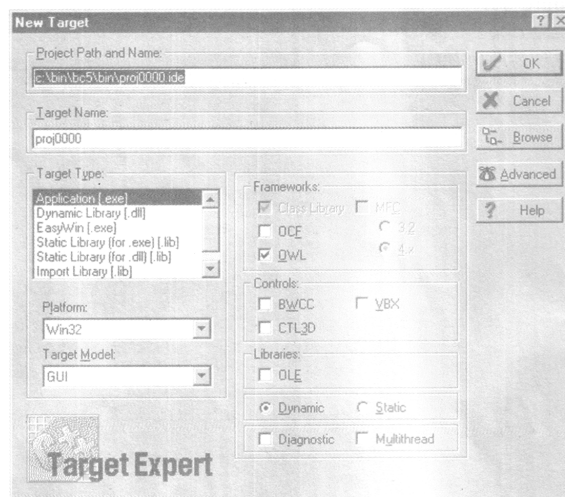



그림 6-4. 완성된 New Target대화칸

6.2.2 대상과제에 원천파일의 추가

대상과제파일을 보관한후 IDE는 그림 6-5에서 보여 준 대상과제파일을 창조한다. 이 창문은 대상과제를 작성한 파일들을 현시한다. 기정설정으로 IDE는 항상 대상과제에 대하여 3개의 파일(Target.Cpp, Target.def, Target.rc)을 추가하는데 여기서 Target는 TargetName마당에 입력된 이름이다. EzWindows응용프로그람에서 .def와 .rc파일들은 제거되여야 한다. 대상과제에서 파일을 제거하기 위해 마우스로 파일을 선택하고 오른쪽마우스단추를 누르면 여러개의 마당으로 이루어 진 차림표가 펼쳐진다. 대상과제에서 파일을 제거하기 위한 Delete node를 선택한다. .def와 .rc파일들을 둘 다 제거하기위해 이와 같은 동작을 반복수행한다.

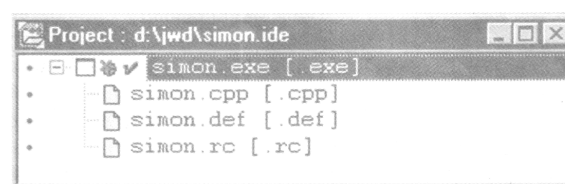



그림 6-5. Borland C++ 대상과제창문

필요 없는 파일들을 삭제한후 대상과제를 작성한 파일들을 추가해야 한다. 그림 6-1에서 보여 준바와 같이 대상과제는 응용프로그람 Simon.cpp와 Simmain.cpp, simobj.cpp, EzWindows서고들을 포함한다. 이와 같이 우에서 지적한 파일들을 추가하여야 한다. 대상과제에 파일을 추가하기 위해 대상과제창문에서 대상과제의 뿌리파일을 선택한다. 실례에서 본 뿌리파일은 simon.exe이다. 뿌리파일을 선택한후 지령차림표를 내보내기 위해서는 마우스를 오른쪽누르기하고 Add node를 선택한다. 이것은 표준파일선택대화칸을 현시한다. 원천파일들의 위치로 이동시키고 조종건을 유지하면서 파일들을 누르기하여 대상과제에 추가하도록 하나씩 선택한다. 대상과제에 대한 추가를 끝마치려면 파일들을 선택한 다음 Open단추를 누른다. 머리부파일들은 추가되지 않는다는것을 알아야 한다. 그림 6-6은 응용프로그람원천파일들이 선택된 파일선택대화칸을 돌려 준다.

대상과제에 Ezwindows서고 ezwin.lib파일도 추가해야 한다. Add node지령을 다시 선택한다. EzWindows서고파일들이 위치를 이동하기전에 서고파일을 현시하기 위하여 File of type마당(그림 6-6)을 변화시켜야 한다. 그러기 위하여 Libraries(*.lib)항을 선택한다. 다음으로 EzWindows서고의 위치에 이동시키고 ezwin.lib를 선택한다. 대상과제에 서고를 추가하기 위해 open단추를 누른다.

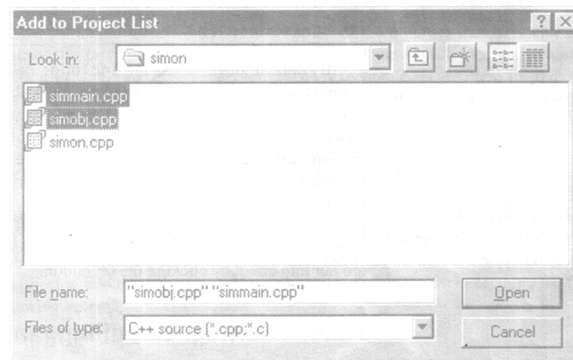



그림 6-6. 파일선택대화칸

6.2.3 대상과제항목의 설정

대상과제파일설치의 마지막단계는 대상과제선택들이 정확하게 설정되는가를 확인하는것이다. EzWindows머리부정의파일들은 물론 체계부파일들까지도 탐색하기 위한 경로들이 설정되는가를 확인해야 한다. 대상과제선택들을 변화시키면 대화칸을 내보내기 위해 다음의 지령을 수행한다.

Option -> Project

이 지령에 의하여 그림 6-7과 같은 대화칸이 나타난다.

기정설정으로 IDE는 체계머리부정의파일이 배치된 경로를 포함하고 있는 Include마당과 체계서고파일들이 배치된 경로를 포함하는 Library마당을 설정한다. Ezwindows응용프로그람을 성공적으로 번역하기 위해서는 Include:항에 EzWindows머리부정의파일들의 위치를 추가해야 한다. 추가등록부들은 마당에 그것들을 삽입하고 반두점으로 항들이 분리되여 추가된다. 실례로 EzWindows머리부정의부파일들의 위치는

D:\ book\ ezwin\ include

이다.

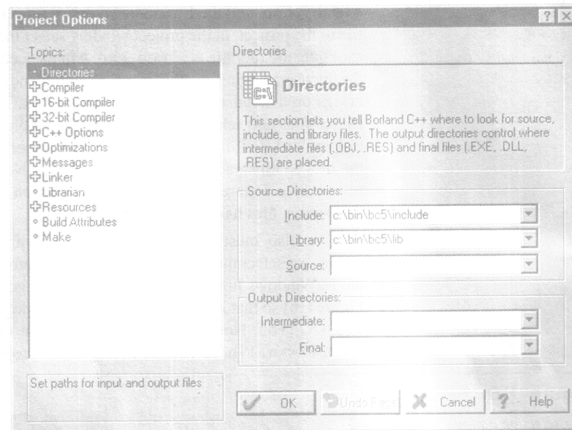



그림 6-7. Borland C++ Project Option대화칸

이 머리부정의등록부항이 추가된후 Project Options대화칸을 그림 6-8에 보여 준다. 항들이 모두 정해 지면 OK단추를 눌러 변화동작을 수행한다.

6.2.4 편집, 콤파일, 련결

대상과제파일이 설치된후 IDE는 프로그람이 쉽게 편집, 콤파일, 련결되도록 한다. 프로그람모듈을 편집하기 위해 간단히 대상과제창에서 파일을 두번 누른다. 이 동작은 편집창에서 파일을 열기한다. 그때 사용자는 종합편집기를 리용하여 파일을 변화시키도록 한다. 자기원판에 변경한 파일을 보관하기 위해 파일보관단추를 누른다. 응용프로그람을 번역하고 련결하기 위하여 다음의 동작들을 집행한다.

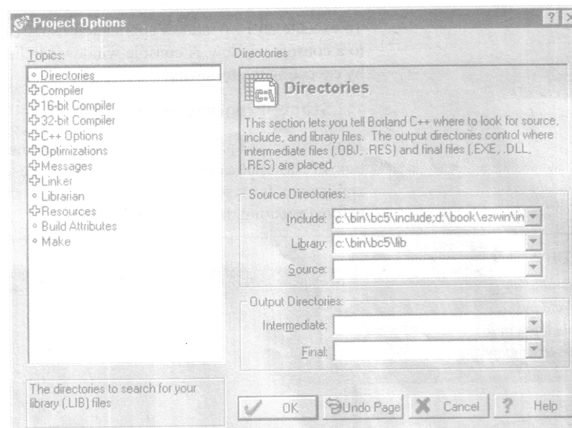



그림 6-8. EzWindows머리부정의등록부를 추가한후 Borland C++의 Project Options대화칸

Project -> Make all

이 지령은 마지막번역후 변화된 모듈들을 모두 번역하고 실행단위안에서 객체파일들과 요구한 서고들을 련결하기 위하여 IDE를 관리한다. 실행파일은 Simon.exe파일로 되는데 그 리유는 대상과제파일을 설치할 때 지정한 이름이기때문이다.

6.2.5 대상과제파일의 보관

대상과제파일을 보관하기 위해 대상과제창문이 능동상태인가를 확인한다. 창문은 대상과제창문안을

누르기하여 능동상태로 만들수 있다. 파란색도구띠를 가진 창문은 마우스초점을 가질수 있다. 대상과제창문이 초점을 맞추면 대상과제파일은 다음의 지령을 실행하여 보관할수 있다.

File -> Save

이미 열려 진 편집창들이 파일을 보관하지 않았다면 모든 능동상태의 파일들은 다음의 지령을 실행하여 보관될수 있다.

File -> Save all

6.2.6 EzWindows프로그람의 실행

Borland C++를 리용하여 EzWindows응용프로그람을 실행하자면 조종창문으로 바꾸어야 한다. 조종창문은 다음의 지령들을 실행하여 창조할수 있다.

Start -> Program -> Command Prompt

조종창문의 지령대기문상태에서 실행할수 있는 응용프로그람의 위치로 등록부들을 변화시킨다. 지령대기상태에서 응용프로그람의 이름을 써넣어서 실행시킨다. 그림 6-9는 Simon응용프로그람을 실행하기전에 조종자를 보여 준다.

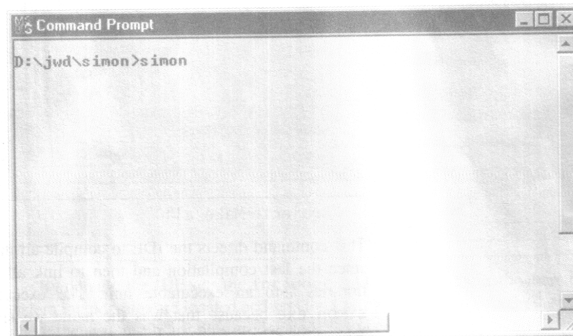



그림 6-9. 조종창문에서 EzWindows응용프로그람의 실행

6.3 Microsoft Visual C++통합개발환경

이 통합개발환경은 대상과제파일에서 응용프로그람을 건설하는 방법에 대한 정보를 제공한다. 여기서 대상과제파일들은 확장자가 .dsp로 된다.이 통합개발환경의 기본창문을 그림 6-7에 보여 준다. 창문의 제일 꼭대기에 가로 지른 띠는 File, edit, Save, View와 같은 여러가지 지령들을 포함하고 있다. 지령항목들을 눌러서 여러가지 차림표가 나타나게 한다. 실례로 Help를 누르면 그림 6-10에 보조차림표가 나타난다. 이 차림표항목밑에 여러개의 단추들이 있다. 이 부분을 도구띠라고 한다. 도구띠는 자주 리용되는 지령들에 대한 지름건을 의미한다. 실례로 왼쪽으로부터 두번째 단추는 파일열기지령을 수행한다. 파일을 열기 위한 또 다른 방법은 아래와 같은 명령을 실행하는것이다.

File -> Open

Microsoft Visual C++ IDE의 기초에 대해 많이 학습한후에 일부 갱신된 환경특성들을 리용해 볼수 있다.

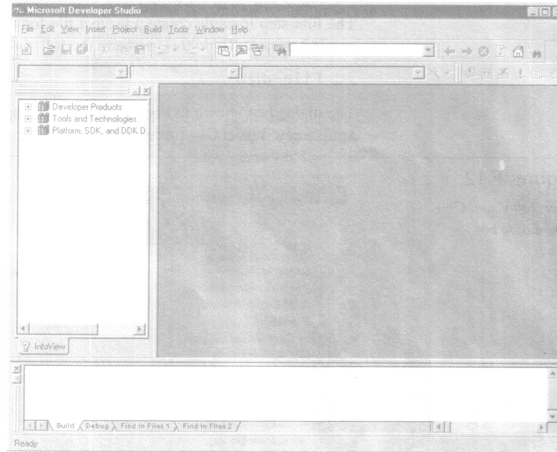



그림 6-10. Microsoft Visual C++ 통합개발환경

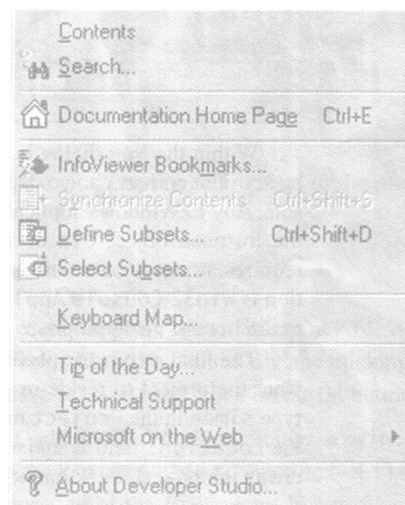



그림 6-11. Microsoft Visual C++도움말차림표

Help지령을 실행하여 통합개발환경의 임의의 특성에 대한 방조를 얻을수 있다.

6.3.1 작업공간의 창조

EzWindow응용프로그람을 위한 대상과제파일을 창조하는 첫 단계는 대상과제를 생성하는것이다. 이를 위해 다음과 같은 동작을 한다.

File -> New

그림 6-12에서 보여 준 대화칸이 현시된다. 이 대화칸은 새로운 Visual C++대상과제를 창조하기 위한것이다.

이 대화칸에서 건설할 응용프로그람의 형태를 설정하여야 한다. 적당한 선택은 응용프로그람이 표준입출력장치를 요구하는가, 하지 않는가에 관계된다. 입출력흐름객체 cin을 통해 사용자로부터 건입력을 접수하거나 cout를 통하여 현시를 진행하는 임의의 EzWindows응용프로그람은 표준입출력장치를 요구한다. 이 형태의 응용프로그람에 대한 정확한 선택은 win32 console application이다. 응용프로그람의 입출력흐름서고를 리용하지 않는다면 이때 win32 application을 선택해야 한다.

마지막단계는 대상과제의 위치를 지정하는것이다. 실례로 대상과제를 D:\ jwd\ simon에 있게 하려고 한다. 그래서 D:\ jwd로 이동하고 대상과제이름마당에 Simon을 써넣는다. 대상과제이름을 써넣을

때 Location마당은 D:¥jwd¥simon으로 되여 자동적으로 갱신된다. 대상과제를 창조하기 위하여 OK단추를 누른다.

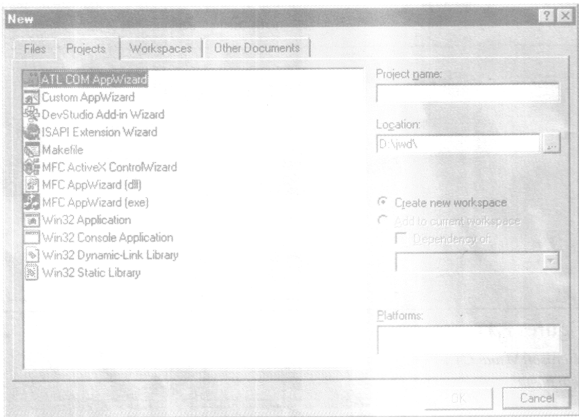



그림 6-12. Microsoft Visual C++의 New대화칸

6.3.2 대상과제에 원천파일추가

대상과제파일을 창조한후 IDE는 그림 6-13에서 보여 준 Class View and FileView창문에 대상과제이름과 대상과제를 구성한 클라스 및 파일들을 써넣는다(그림 6-13).

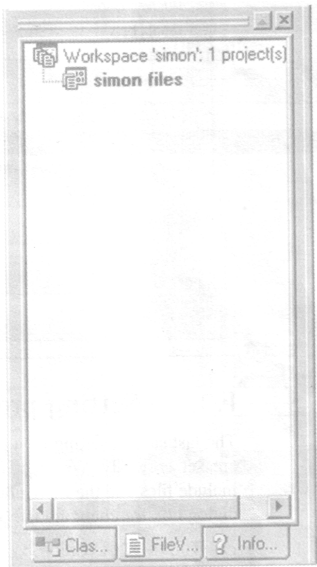



그림 6-13. Microsoft Visual C++ ClassView와 FileView창문

대상과제에 파일들을 추가하는 대화칸을 현시하기 위해서는 Project -> Add to Project -> files지령을 실행해야 한다. 이 동작이 수행되면 그림 6-14와 같은 대화칸이 현시된다. Ctrl건을 누르고 원천파일들을 누르기하여 대상과제에 여러개의 파일들을 추가할수 있다. 파일이 모두 선택된 다음 OK단추를 누른다.

또한 대상과제에 EzWindows서고 ezwin.lib파일도 추가해야 한다. 지령 Project -> Add to Project -> files를 수행한다. EzWindows서고파일들이 위치를 이동시키기전에 서고파일들을 현시하기 위하여 Files of Type마당을 변화시켜야 한다. 그러기 위하여 Library Files(.lib)항을 선택한다. 그다음 EzWindows서고의 위치를 이동시키고 ezwin을 선택한다. 대상과제에 서고를 추가하기 위해 OK단

추를 누른다.

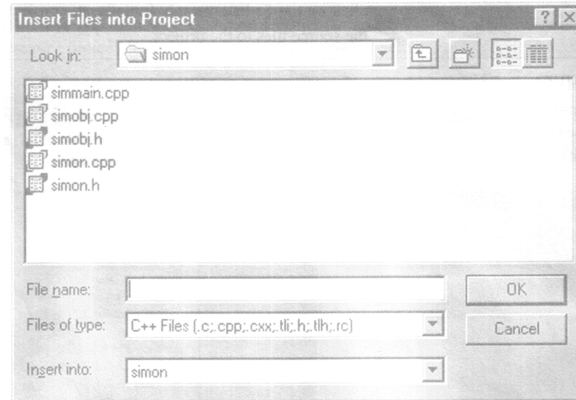



그림 6-14. Microsoft Visual C++ Insert Files into Project대화칸

6.3.3 대상과제선택항목들의 설정

대상과제파일을 설치하는 마지막단계는 대상과제선택들이 정확히 설정되였는가를 확인하는것이다. 체계머리부정의파일과 Ezwindows정의부머리부파일들을 탐색하기 위한 경로가 정확히 설정되였는가를 확인해야 한다. 대상과제선택을 변화시키기 위한 대화칸은 다음의 동작을 수행하여 현시할수 있다.

Tools -> Options

이 동작이 수행되면 그림 6-15와 같은 대화칸이 현시된다.

기정설정에 의하여 IDE는 체계머리부정의파일들이 배치된 경로를 포함하는 Directories항을 설정한다. Ezwindows응용프로그람을 성과적으로 번역하기 위해서는 머리부정의파일들을 탐색하기 위한 등록부묶음들에 EzWindows정의파일들의 위치를 추가해 주어야 한다. 추가등록부들은 공백건을 누르고 그 경로를 써넣는것으로서 추가할수 있다. 례를 들어 EzWindows머리부정의파일들의 위치는 D:¥book¥ezwin¥include이다. 이 머리부정의파일들에 대한 경로를 추가하는것은 가능하다. 다음단계는 Option파일대화칸을 제시한다. 항들이 모두 만들어 지면 OK

단추를 눌러서 변화된 효과들이 가능하게 한다.

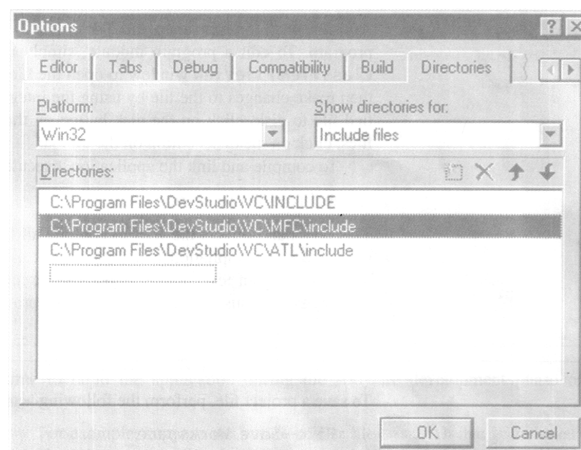



그림 6-15. Microsoft Visual C++ Options대화칸

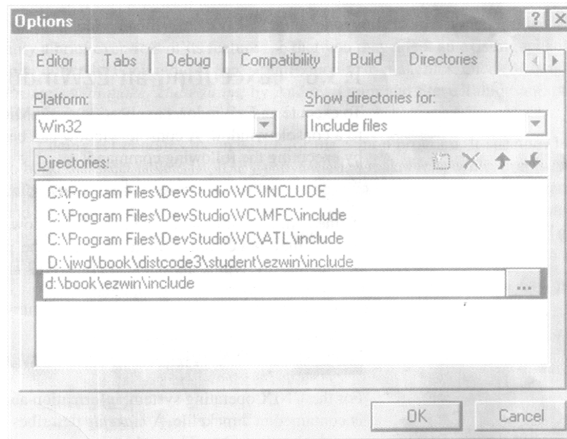



그림 6-16. EzWindows 머리부정의등록부를 추가한후 Microsoft Visual C++ Options대화칸

6.3.4 편집, 콤파일, 련결

대상과제파일이 설치된후 IED는 프로그람이 쉽게 편집, 콤파일, 련결되도록 한다. 프로그람모듈을 편집하자면 편집창에서 파일을 열기하여 class view/file view창문에서 파일을 두번 누른다. 이때 사용자는 종합편집기를 리용하여 파일을 변화시켜야 한다. 자기원판에 변경된 파일을 보관하기 위하여 도구띠의 파일보관단추를 누른다. 변경된 모든 파일들을 보관하자면 도구띠의 파일보관단추들을 누른다. 응용프로그람을 번역하고 련결하기 위해서는 다음의 지령을 수행해야 한다.

Build -> Build simon.exe

이 지령은 마지막콤파일을 진행한후 변화된 모듈들을 모두 콤파일한다. 그다음 객체파일들과 요구되는 서고들을 실행묶음으로 련결하기 위하여 IDE를 조종한다. 실행파일은 Simon.exe로 되는데 그 리유는 그것이 대상과제의 이름이기때문이다.

6.3.5 대상과제파일의 보관

대상과제파일을 보관하기 위해서는 다음의 동작을 수행해야 한다.

File -> Save WorkSpace

열려 진 임의의 편집창문들이 파일을 보관하지 않고 있다면 모든 능동상태의 파일은 다음의 지령을 실행하여 보관할수 있다.

File -> Save All

6.3.6 EzWindows응용프로그람의 실행

Microsoft C++를 리용하는 EzWindows응용프로그람을 실행하기 위해서는 표준입출력창문으로 바꾸어야 한다. 이 창문은 다음의 지령으로써 현시할수 있다.

Start -> Program -> Command Prompt

표준입출력창문의 지령대기상태에서 응용프로그람이 실행될수 있다. 이때 응용프로그람의 이름을 써넣어야 실행된다. Simon응용프로그람을 실행하기전의 표준입출력장치를 그림 6-17에 보여 준다.

6.4 UNIX제작파일들

UNIX조작체계에서 응용프로그람을 건설하기 위한 방법에 대한 정보는 제작파일에 포함된다. 제작파일은 응용프로그람들이 어느 파일에 의존하는가를 서술한다. make프로그람은 어느파일들이 다시 번역되고 응용프로그람의 실행단위를 만들기 위하여 련결되는가를 결정하는 정보를 리용한다. 기초개념은 make프로그람이 변했거나 변화된 파일에 의존하는 파일들만 콤파일한다는것이다. 이 방법은 대상과제의 쏘프트웨어개발기간에 응용프로그람을 건설하는 시간을 줄인다.

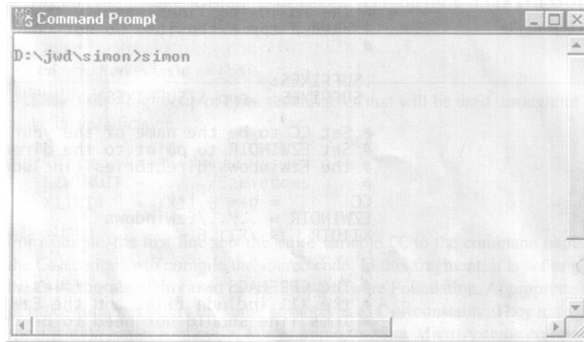



그림 6-17. 표준입출력창문에서 EzWindows응용프로그람의 실행

례를 들어 Simon.cpp나 Simobj.cpp파일까지 포함한 머리부파일 Simon.h가 있다고 하자. 그러나 이 파일들에는 Simmain.cpp파일이 포함되지 않는다. Simon.h파일이 변경되면 Simon.cpp와 Simobj.cpp파일들은 다시 번역되여야 한다. 그러나 Simmain.cpp파일은 그렇지 않다. UNIX의 제작환경은 아주 위력하다. 이 부록에서는 사용자의 요구에 현존제작파일들을 호환시킬수 있도록 그 특성들에 대한 간단한 견해만 주게 된다.

목록 6-1은 Simon프로그람을 위한 제작프로그람을 소개하였다. Simon을 실행할수 있도록 건설하는 방법도 소개하였다. 제작파일에 대한 설명서는 일반적으로 make라는 파일에 기억된다. 제작프로그람을 실행하고 응용프로그람을 만들기 위하여 입력제촉문에 지령 make를 써넣어야 한다. make프로그람은 make파일이나 make파일로 이름 지어 진 파일에 대한 작업등록부에 가서 그것을 처리한다. 제작파일에서 주해는 #기호로 시작한다.

목록 6-1. Simon제작파일

```
#
# 다음의 행은 C++파일이 cpp확장자를 가진다는것을 설명한다.
# 이 행을 변화시키지 마시오.
#
.SUFFINXES:
.SUFFINXES:  .cpp $(SUFFINXES)
# C++콤파일러는 확장자가 cc라는 이름으로 설정한다.
# EzWindows등록부들을 포함하는 서고를 지적하기 위하여 EzWindir #를 설정한다.
CC          = g++
EZWINDIR = ../../EzWindows
X11DIR = /X11.6
#
# CPPFLAGS변수는 X11이 어디서 파일을 포함하는가를 지정한다.
# 이 행은 변하지 말아야 한다.
```

```
CPPFLAGE=-I$(X11DIR)include - I$(EZWINDIR)/include
#LDFLAGS변수는 서고파일을 어디서 찾는가를 지정한다. 이 행은 변화되지 말아야 한다.
LDFLAGS=-L$(X11DIR)/lib - 1X11 -1socket ￥
-L$(EZWINDIR)/lib -1ezwin -1Xpm
#OBJS변수는 응용프로그람을 건설하기 위해 창조되는 필요한 객체를 번역한다.
OBJS=simmain.o simpbj.o simon.o
#다음의 규칙은 실행성프로그람을 구축하는 방법을 보여 준다.
Simon: $(OBJS)
    $(CC) -o simon $(OBJS) $(LDFLAGS)
simon.o: simon.h
simon.o: simon.h
simobj.o: simobj.h
#프로그람이 확장자 cpp를 가진 프로그람을 처리하는 방법을 보여 준다.
#대체로 cpp확장자가 정의되지 않으므로 이 행이 필요하다.
.cpp.o:
    $(CC) $(CPPFLAGS) -c $<
#표준련습으로서 제거목적은 대부분의 제작파일에 포함된다.
#실행파일은 제거되며 모든 객체파일을 삭제하고 실행된다.
clean:
    rm -f *.o simon
```

C++로 프로그람을 작성하기때문에 주해는 프로그람작성공정을 설명하는 문자를 써넣음으로써 쉽게 프로그람을 리해할수 있게 한다.

제작파일에서 다음과 같은 형들은 C++파일들이 확장자 .cpp를 가진 make프로그람이라는것을 의미한다.

```
.SUFFIXES:
.SUFFIXES:  .CPP $(SUFFIXES)
```

일부 C++체계들은 확장자가 .cc인 C++파일들을 제공한다. 만일 이것을 체계상에서 동작시키려면 사용자는 두번째 행에 있는 .cpp를 .cc로 변화시켜야 한다. 이 책에서 취급하는 C++파일들은 확장자가 .cpp이다.

다음의 행묶음은 제작파일설계에서 계속 리용되는 변수들을 설정한다.

```
CC        = g++
EZWINDIR = ../../EzWindows
X11DIR = /X11.6
```

례를 들어 두번째 행은 원천코드를 번역하는 C++콤파일러의 지령이름에 make변수 CC를 설정한다. 이 부분에서는 C++콤파일러가 Freesoft foundation에 의하여 할당된 g++로 된다. make변수들에 대한 합리적인 리용은 C++상수와 같이 편리성을 제공한다. 설계공정을 더 쉽게 해보자. 실례로 Solaris조작체계로 기동하는 싼콤퓨터상에서 C++콤파일러를 호출하기 위한 지령은 CC이다. 이 콤파일러는 변수 CC to CC를 설정하여 리용할수 있다. 두번째 행은 EzWindows머리부정의등록부와 서고등록부들을 포함하는 경우에 make변수 EZWINDIR를 설정한다.

이 변수들은 개별적사용자체계에 대응하는 쏘프트웨어의 위치를 반영하도록 변화시켜야 한다.

```
CPPFLAGS = -I$(X11DIR) / include -I$(EZWINDIR)/include
```

이 행에서는 X11과 EzWindows머리부정의파일들이 어디에 있는가를 C++콤파일러에 알려 주는 변수를 정의한다. 이 행은 변경할 필요가 없다. 이와 류사하게 다음의 값주기는 필요한 서고파일들을 어디서 찾는가를 UNIX적재프로그람 1d에 알려 주는 변수를 설정한다.

```
LDFLAGS =-I$(X11DIR)/lib -1X11 -1socket \
-L$(EZWINDIR)/lib -1ewin -1Xpm
```

대부분의 환경에서 이 행은 변경을 요구하지 않는다.

```
OBJS= simmain .o simobj.o simon.o
```

이 명령문은 응용프로그람을 작성한후 목적파일들의 이름들에 변수 OBJS를 설정한다. Simon응용프로그람은 3개의 응용프로그람모듈 즉 simmain.o, simobj.o, simmon.o를 가지고 있다. 서로 다른 응용프로그람에 대해서는 이 행에서 simmon응용프로그람을 작성한후 목적모듈에 OBJS를 설정한것이 변경되여야 한다. 다음의 행들의 묶음이 제작파일의 핵으로 된다. 이 행들은 응용프로그람을 작성하는 방법을 규정하는 종속규칙들이다. 종속규칙은 다음과 같은 형태를 가진다.

```
Dependency line
command line
```

종속행은 목적(target)이 의존하는 파일을 지정한다. 그 파일이 목적에 의하여 어떤 방법으로 만들어 질 때 목적은 파일에 의존한다. 가장 일반적으로 리용되는 종속행의 형태는

```
target: dependents
```

이다. target는 만들어 지는 파일의 이름과 확장자이다. target이름에서는 공간이나 타브들을 첫 머리에 놓을수 없다. 종속들은 목적보다 더 새로운것인가 아닌가를 보기 위하여 변경날자와 시간, make가 검사하는 파일들이다. 매 종속파일들은 공백을 첫 머리에 놓아야 한다.

지령행은 목적을 만드는 방법을 보여 준다. 지령은 제일 마지막 한개의 공간이나 타브가 안으로 약간 들어 가게 하여야 한다. 그밖의 경우에는 target로 해석된다. 실례로 simmon제작파일에서

```
simon: $(OBJS)
$(CC) -o  simon $(OBJS)  $(LDFLAGS)
```

은 목적 simon이 simmain.o, simobj.o, simmon.o값을 가지는 OBJS에 의존한다는것을 지정한다. 만일 이 객체파일가운데서 임의의것이 목적 Simon보다 더 새로운것이라면 제작파일은 지령행을 실행한다. 지령행은 객체파일보다 더 새로운 Simon을 만든다. 만일 객체파일들이 Simon보다 낡은것이라면 Simon은 최신으로 되며 비능동상태가 되여야 한다.

make프로그람은 파도종속파일들을 처리한다. 즉 종속파일이 목적으로서 다른 경우에 제작파일에 나타나면 make는 본래의 목적에서 종속파일을 리용하기전의 목적을 갱신하거나 창조한다. 례를 들면 파일

```
simmain.o:  simon.h
```

은 simmain.o이 머리부파일 simon.h에 의존한다는것을 지정한다. 그래서 simonb.h를 편집하고 그 변화들을 보관하면 그 시간과 날자형은 simmain.o보다 더 늦어 진다. 만일 simon을 만드는 make지령을 발생시키면 제작파일은 simon이 simon.h에 의존하는 simmain.o에 의존한다고 본다. Simon.h가 simmain.o보다 더 새로운것이라는것으로부터 제작파일은 우선 simmain.o을 만들려고 한다. 제작파일이 simmain.o을 만드는 방법이 지정하는 지령행을 가지지 않을수 있지만 그것은 지령행을 가진다.

```
.cpp.o:
$(CC) $(CPPFLAGS) -c  $<
```

이 행은 함축된 종속규칙이다. 함축된 종속규칙은 파일의 한 형태가 다른 형태의 파일로부터 구축된다는것을 지정하며 구축방법을 서술한다. 웃행은 .cpp파일로부터 .o파일들이 구축되며 .o파일들이 C++

콤파일러에 의해 구축된다는것을 지정한다. 그러므로 make가 simmain.o를 작성하기 위하여 simmain.cpp를 번역한다는것을 알게 된다. 간단하게 make는 simon.cpp와 simobj.cpp를 각각 번역하여 simon.o와 simobj.o를 만든다. 필요한 목적파일이 작성된후 그 시간과 날자형들은 simon보다 더 늦을것이다. 따라서 simon을 작성하는 지령이 실행된다.

만일 인수가 없는 지령 make가 입력제촉상태에서 입력되면 make는 첫번째 종속규칙의 지정한 목적을 만들려고 한다. w는 또한 개별적인 목적을 지정할수 있다. 지령

```
make   simmain.o
```

은 simmain.o과 그의 모든 종속파일들을 작성한다. make의 이러한 리용은 단위모듈을 손쉽게 강제번역할수 있게 한다. make프로그람은 많은 선택항목들을 가진다. 한가지 유효한 선택항목은 -n이다. 지령

```
make -n
```

은 make지령들을 인쇄는 하지만 수행하지는 않는다. 이 선택항목은 제작파일의 오유를 수정하는데 효과적이다. 지령

```
make -B
```

는 모든 목적들을 시간과 날자형에 관계없이 건설한다. 이 선택항목은 make를 실행시켜 스크래치파일로부터 모든것을 다시 건설하는데서 유용하다. 모든 선택항목들에 대한 목록은 쉘입력재촉상태에서 다음의 지령을 타자하여 보여 줄수 있다.

```
man make
```

6.4.1 EzWindows응용프로그람의 실행

X11창문체계를 가지는 UNIX에서 EzWindows응용프로그람을 실행시키려면 쉘을 실행시켜 말단창문(xterm)을 창조해야 한다. 말단쉘을 창조하는것은 사용자가 리용하고 있는 창문관리기에 의존된다. 대부분의 창문관리기에서는 화면의 빈 령역으로 마우스를 움직이고 오른쪽누르기하면 xterm을 기동하게 하는 차림표가 나타난다.

그림 6-18은 대표적인 xterm창문을 보여 준다. 쉘입력재촉상태에서 EzWindows응용프로그람의 이름을 써넣어서 실행시킨다.

그림 6-18. UNIX에서 X11 xterm

색 인